UNIVERSIDAD AUTÓNOMA DEL ESTADO DE HIDALGO

ESCUELA SUPERIOR DE HUEJUTLA

LICENCIATURA EN CIENCIAS COMPUTACIONALES

PROYECTO:

DISEÑO DE SISTEMA DE CONTROL Y GESTIÓN DE PROVEEDORES Y AGENTES PARA EMPRESA INMOBILIARIA

PRESENTA:

JOSÉ ESCUDERO CISNEROS

ASESORES:

MTRO. VÍCTOR TOMÁS TOMÁS MARIANO

MTRO. FELIPE DE JESÚS NÚÑEZ CÁRDENAS

Noviembre 2017

Índice

Índice de Ilustraciones

Índice de Tablas

Resumen

El presente trabajo tiene como propósito principal implementar un sistema de información automatizado que optimice la gestión de los procesos administrativos de las siguientes áreas: suministros, cuentas por pagar, proyectos de obra y contabilidad de la empresa inmobiliaria. Este software permite controlar cada uno de los procesos administrativos que allí se realizan, los cuales involucran: registro de proveedores, validación de los proveedores, registro de órdenes de compra, validación de órdenes de compra, entre otros. Con este sistema se automatizó los procesos operativos y se suministró una plataforma de información necesaria para la toma de decisiones aportando información confiable, precisa y adecuada que contribuye a minimizar los riesgos y generar procesos más eficaces. Dicho trabajo siguió la metodología XP, la cual permite crear una solución, apoyada en el uso de fases y herramientas sustentadas para modificar una situación; la arquitectura que se usó fue la MVC, con el objetivo de contar con un modelo que facilite la consulta y manejo del código. Para la creación del sistema se utilizaron herramientas tales como: SQL Server. Visual Studio 2017, JavaScript, JQuery.

Palabras Claves: Metodología XP, Arquitectura MVC, Sistema de Información, Plataforma.

Abstract

The main purpose of this work is to implement an automated information system that optimizes the management of administrative processes in the following areas: supplies, accounts payable, construction projects and accounting of the real estate company. This software allows to control each one of the administrative processes carried out there, which involve: supplier registration, validation of suppliers, purchase order registration, validation of purchase orders, among others. With this system, the operating processes were automated and an information platform was provided, necessary for decision-making, providing reliable, accurate and adequate information that helps minimize risks and generate more efficient processes. This work followed the XP methodology, which allows creating a solution, supported by the use of phases and supported tools to modify a situation; the architecture that was used was the MVC, with the objective of having a model that facilitates the consultation and management of the code. For the creation of the system, tools such as: SQL Server were used. Visual Studio 2017, JavaScript, JQuery.

Keywords: XP Methodology, MVC Architecture, Information System, Platform.

I. Generalidades

1.1 Introducción

Los avances científicos y tecnológicos han influido en la vida cotidiana de la humanidad, las personas buscan constantemente la automatización de sus actividades y labores, con la finalidad de que estas se realicen con mayor calidad y menor tiempo. En la actualidad el Internet y muchas herramientas que trabajan de la mano se han convertido en una necesidad para las empresas, porque se ven obligadas a una constante actualización de las tecnologías que implementan, como beneficio obtienen la mejora de sus procesos y actividades.

Los lenguajes de programación y otras herramientas que trabajan en conjunto a menudo son utilizados por las empresas como un medio para lograr la automatización de sus procesos y cumplir con sus objetivos, con el fin de que la información esté disponible en el momento que se requiera y para el procesamiento que se le desea aplicar. Para ello es necesario la utilización de sistemas de información los cuales ayuden a procesar dicha información y con ello a optimizar las actividades que la empresa demanda.

Los sistemas de información se han utilizado como un medio para tratar de forma sensible, segura, fácil y rápida los datos que se generan y procesan en una empresa u organización, estos ofrecen una manera correcta de tratar a la información, con la salvedad de que los usuarios estén conformes de utilizarlo.

La presente documentación describe el desarrollo de un sistema de información para mejorar los procesos de gestión de una empresa inmobiliaria. El sistema está desarrollado con la tecnología necesaria para cumplir con los objetivos, algunas de estas son SQL Server para el

alojamiento de la base de datos, Visual Studio, ASP Net Core, Bootstrap, JavaScript, para la generación de vistas y comunicación con la base de datos, entre otras.

1.2 Planteamiento del problema

Para estar al margen del mundo actual es necesario ajustarse al desarrollo y crecimiento del ámbito tecnológico, lo cual sirve como mecanismo para acceder a la información, cumpliendo ciertos parámetros como, por ejemplo, rapidez, privacidad, confidencialidad, entre otras; tales que contribuyan al desarrollo de las empresas. Lo antes mencionado es considerado por las empresas públicas o privadas, como herramienta para la optimización de sus procesos. Desde este punto la implementación de sistemas de información o sistemas automatizados contribuyen como una alternativa óptima y eficaz para mejorar los procesos de gestión y así mismo lograr un mejor desempeño laboral.

Actualmente los sistemas de información tienen un rol sumamente importante en el ámbito de las empresas, porque tienen la finalidad de poder hacer frente a las necesidades que demande su entorno y con ello poder subsistir en el mundo globalizado. Por ello es de suma importancia que las empresas obtén por usar los sistemas de información desarrollados con la mejor tecnología posible y con ello se dé solución a los inconvenientes que se estén presentando.

Considerando lo mencionado en el párrafo anterior, para lograr desarrollar un sistema de información es importante decidir de la manera más correcta, la metodología, la arquitectura y las tecnologías que se van a implementar, así como también considerar las reglas del negocio, y con ello lograr los objetivos requeridos. Un sistema de información necesita de una base de datos la cual sirva para alojar la información necesaria, así como también de formularios o vistas con las cuales los usuarios puedan interactuar y poder realizar sus actividades de una mejora manera.

Con base a lo anterior, se desea desarrollar un sistema para una empresa inmobiliaria que permita a los usuarios recolectar información, almacenarla y procesarla de una manera más fácil, rápida, pero sobre todo eficaz. Con el sistema se pretende dar solución a los inconvenientes que la empresa presenta y mejorar los procesos de gestión de las áreas involucradas.

1.3 Justificación

Este proyecto implementa una forma de automatizar y mejorar los procesos de gestión de una empresa inmobiliaria, mediante la utilización de las tecnologías más sofisticadas y que están al alcance de la misma, con la finalidad de mejorar la calidad de cada una de las actividades que se realizan las áreas involucradas de la empresa y así poder contribuir al óptimo funcionamiento de la misma.

El sistema será capaz de recolectar información introducida por los usuarios a través de las diferentes vistas, almacenarla en una base de datos relacional; para que posteriormente sea manipulada y sometida a distintas etapas que son necesarias para el cumplimiento de los objetivos y metas que se desean alcanzar. Esta herramienta de automatización dotará al personal que labora en las áreas involucradas de un recurso tecnológico que facilite el desempeño de sus labores y minimice la duplicidad de trabajo, permitiendo la agilización de los procesos. Para la empresa, el acceder a las capacidades que ofrece un sistema de información es una importante ventaja competitiva.

Este sistema puede servir de ejemplo para desarrollar otros sistemas enfocados a distintos ambientes y sectores productivos, porque la funcionalidad puede ser similar en muchos casos. Por ejemplo, en todos los sistemas de información se generan datos de distintos tipos, se procesan de la manera más adecuada para obtener lo requerido. Para ello se emplean distintas herramientas que

están al alcance de la empresa u organización pero que así mismo den la mejor solución posible y esta pueda mantenerse por un tiempo considerable.

1.4 Objetivo general

Analizar, diseñar e implementar un sistema de información web automatizado para una empresa inmobiliaria con el fin de optimizar la gestión de cada uno de los procesos de las áreas involucradas, mediante la arquitectura MVC.

1.5 Objetivos específicos

- Estudiar el modelo de negocio del (las) área (áreas) involucradas, para obtener una visión profunda a nivel conceptual.
- Determinar los requisitos funcionales, no funcionales del sistema o interfaz, para fortalecer y mejorar el modelo actual.
- Diseñar y construir la arquitectura que debe tener el sistema a desarrollar considerando todos los requisitos.
- Implementar el sistema, poniéndolo en marcha dentro de su plataforma de operación.

II. Estado del arte

En este apartado se abordarán algunos antecedentes relacionados con el tema en cuestión, para ello se procedió a la revisión de algunos estudios que incorporan elementos de relevancia. Entre ellos:

2.1 Implementación de los sistemas de información en las empresas inmobiliarias

Ruiz J. (2009) desarrolló un proyecto denominado: sistema de gestión de una inmobiliaria, el cual fue presentado en la Universidad Pontificia Comillas Escuela Superior de Ingeniería (ICAI). El objetivo de este proyecto fue el estudio, definición y desarrollo de una herramienta fácil de entender y usar, que permite controlar y gestionar las actividades de la empresa. Para su desarrollo se utilizó el gestor de base de datos MySQL, Eclipse, Adobe Dreamweaver CS3, Apache, Tomcat, Abobe Photoshop CS3, JavaScript, Java EE 5, Ajax, CSS, entre otras herramientas. El sistema dio solución a la mayoría de problemas, relacionados con la no automatización de procesos, que generaban esa ineficiencia en la empresa.

El trabajo realizado por Jiménez H. (2013) titulado: Integración de Información Geográfico para el Análisis Inmobiliario, fue presentado en la Universidad de Sonora. El objetivo era integrar un sistema de información geográfica con la información inmobiliaria necesaria para la realización de análisis y mediciones para establecer las relaciones que definen el valor del suelo urbano en la Ciudad de Hermosillo, Sonora, México. Información en formatos y archivos de fácil utilización con herramientas de análisis estadístico ordinal y estadístico espacial. Se utilizaron las herramientas de ArcGIS Desktop (ArcMap, ArcGlobe, ArcCatalog, MXD Doctor, entre otras), las cuales contienen aplicaciones que permite crear, analizar, almacenar y difundir datos, modelos, mapas y globos en 3D, para la gestión de las bases de datos de información atribuida a los vectores geográficos, se utilizó el software de versión libre Apache Open Office 3. Como herramientas

secundarias para el manejo de la información geográfica se utilizó el software de versión libre Mapa Digital v 5.1.0. Este sistema presenta características inmobiliarias y urbanas importantes para el análisis geográfico de diferentes temas de interés, las cuales pueden ser analizadas por herramientas estadísticas y ser representadas en gráficas, mapas temáticos o modelos ráster, para diferentes áreas de análisis, como por ejemplo los AGEB, colonias, manzanas, sectores, divisiones catastrales o alguna sectorización particular.

Centeno J. (2014) desarrolló el proyecto que lleva por nombre: sistema de gestión del proceso de valuación de inmuebles (caso: Integra Ingeniería Inmobiliaria S.A. de C.V.), fue presentado en el Instituto Politécnico Nacional para obtener el Grado de Maestría en Ciencias en Informática, cuyo objetivo fue integrar, en una sola herramienta Web, un sistema que permita llevar el proceso completo de valuación de inmuebles, desde que un cliente solicita un trabajo de avaluó, hasta que es reportado dicha valuación a entidades gubernamentales. Para el desarrollo del sistema usó la metodología XP y la arquitectura MVC. Las herramientas de desarrollo fueron las siguientes: como contenedor servidor de aplicaciones, se usó Tomcat 7 de Apache, como motor de base de datos, se empleó MySQL. También se usó Google Maps, Google Street View, Sencha EXTJS, Servicios Web con protocolo SOAP, Java como lenguaje de programación, usando 3 populares frameworks encargados de resolver la arquitectura MVC y la persistencia a la base de datos, estos frameworks son GXT para la interfaz de usuario, Spring MVC e Hibernate.

El trabajo de Grado realizado por Idarraga J. (2015) titulado: SIG como herramienta estratégica para el sector inmobiliario en la ciudad de Manizales, fue presentado en la Universidad de Manizales para optar al título de Especialista en Sistemas de Información Geográfica. El objetivo de este proyecto fue crear un sistema de información geográfica que sirva como herramienta estratégica en el sector inmobiliario, enfocado a obtener la mejor opción de ubicación

de un inmueble para la compra o renta en la ciudad de Manizales. El SIG está conformado por entidades geográficas que representan el sistema vial de Manizales según el POT, puntos de interés POI de la ciudad y áreas de riesgos geológicos por deslizamiento e inundación. El sistema obtenido sirvió para obtener la mejor ubicación para la compra o renta de un inmueble en la ciudad de Manizales; para escoger la mejor opción de renta, el SIG ofrece varias opciones de inmuebles que concuerdan con las necesidades del usuario, satisfaciendo de esta manera necesidades no solo de ubicación sino también por las características de la vivienda.

2.2 Implementación de la arquitectura MVC en los sistemas de información

Hernández H. (2013) efectúo un proyecto titulado: sistema de información para el control de expedientes clínicos para médicos veterinarios, presentado en la Universidad Central "Marta Abreu" de Las Villas, cuyo objetivo fue la creación de un sistema capaz de llevar un registro histórico de todos los expedientes clínicos, lo cual permitiera la realización de estudios estadísticos para una mejor comprensión de la evolución y adaptabilidad de los diferentes tipos de enfermedades. Además, que sirviera de apoyo a la docencia y facilite el trabajo del médico veterinario. Se usó el patrón de diseño Modelo Vista Controlador (MVC) y las tecnologías MySql, Servidor de aplicaciones Apache que soporte tecnología PPP5 (XAMP o WAMP), JavaScript, entre otras.

El trabajo realizado por Sarmiento C. (2012) titulado: sistema de registro de participantes en eventos académicos. Dicho trabajo fue efectuado para obtener el título de Ingeniero en Computación, el cual tenía como objetivo llevar un control de los participantes que se registran en uno o más de los eventos que organiza la Universidad Simón Bolívar. Para ello se utilizó de la metodología Open Unified Process (OpenUP) del Eclipse Process Framework y consistió en la

concepción, elaboración, construcción y transición de una plataforma web utilizando la arquitectura MVC, ASP.NET y jQuery.

El software resultante permitió a la compañía registrar eficientemente a los participantes en cada uno de los eventos que se organizan, a través de una interfaz web, llevando un control claro de los ingresos y egresos monetarios y de la información asociada al proceso. Además, permitió la manipulación rápida, por medio de reportes y tener el control de las operaciones que realizan los empleados al registrar participantes y servir como un repositorio histórico de los eventos.

Cárdenas C. (2015) desarrolló un proyecto titulado: sistema de comunicaciones internas de la Escuela de Ingeniería en Computación (CI-IC), este proyecto fue presentado en la Universidad Austral de Chile para optar al título de Ingeniero en Computación, y tuvo como objetivo construir un sistema que permitiera la gestión automática de Documentos de Comunicación Interna dentro de la Escuela de Ingeniería en Computación de la Universidad Austral de Chile, Sede Puerto Montt. Para el desarrollo del proyecto se usó la metodología XP (Extreme Programming) y la arquitectura empleada fue MVC. Se usaron las tecnologías ASP.NET, HTML5, PHP, CSS3, JavaScript, JQuery entre otras; las cuales contribuyeron para dar una solución en el menor tiempo posible y con todos los requerimientos funcionales del Sistema.

El trabajo realizado por Gualteros L. (2016) titulado: sistema de información web para la gestión de los recursos bibliográficos en la biblioteca de la IED El Rodeo, fue presentado en la Universidad Distrital de Francisco José de Caldas para obtener el título de Tecnólogo en Sistematización de Datos, el cual tenía como objetivo analizar, diseñar e implementar un sistema de información web para la administración de los recursos bibliográficos de la IED El Rodeo. La metodología usada fue RUP, para el desarrollo se tomó en cuenta el patrón de arquitectura Modelo

Vista Controlador, así como también otras tecnologías. Las tecnologías para el desarrollo fueron PHP, HTML, JavaScript, Jquery, Css, Less, MySql para la base de datos e igual se usó UML para el modelado. El sistema permitió tener un control de los usuarios registrados mediante el uso de las sesiones; que según el Rol habilita las opciones de consulta, solicitud, devolución, registro, eliminación y actualización de los recursos bibliotecarios disponibles en la institución. Permitió tener informada a la comunidad educativa mediante noticias y eventos publicados.

Franco A. (2013) presentó en la Universidad Técnica del Norte sede en Ecuador, el proyecto denominado: aplicación web para la administración online de citas médicas en el Centro Médico de Orientación y Planificación Familiar CEMOPLAF-OTAVALO; utilizando el patrón de arquitectura MVC en PHP. El objetivo del Sistema fue contar con una herramienta informática eficiente que permitiera la obtención de información verídica y actualizada para un funcionamiento acorde a las necesidades de las personas que interactúan en el centro médico. El desarrollo de la aplicación web, se basó en la metodología RUP, se utilizó esta metodología por su simplicidad, comunicación y la realimentación o reutilización del código desarrollado. Las tecnologías utilizadas para el desarrollo del sistema son PHP como lenguaje de servidor, bajo una arquitectura MVC, MySQL como servidor de base de datos, JavaScript como lenguaje de cliente y AJAX como intermediario entre el cliente y los datos del servidor, CodeIgniter como Framework. La Aplicación web logró realizar de forma rápida, transparente y segura la administración de citas médicas al gestionar correctamente las diferentes cuentas de usuario según los perfiles establecidos incrementando la seguridad e integridad de la información.

2.3 Implementación de SQL en los sistemas de información

Choque J. y Nazar F. (2015) llevaron a cabo un proyecto titulado: diseño e implementación del sistema de gestión web que integra componentes de los sílabos de cursos aplicando Laravel en

el proceso de control de avance silábico de la Universidad Privada de Tacna – 2015, cuyo objetivo fue desarrollar una aplicación web que además de automatizar la planificación y control del avance silábico, también formará parte con el sistema principal de la Universidad Privada de Tacna, el mismo que permitiera la interacción con los docentes y de esa manera obtener datos históricos para un mejor planteamiento y control.

El sistema fue diseñado con herramientas libres y gratuitas tanto para la programación como para la funcionalidad. Se usó SQL como gestor de base de datos, Jet Brains PhpStrom como herramienta de programación, Laravel como framework, PHP como lenguaje de programación, Bootstrap, JavaScript, Apache, Windows Server como SO servidor web y Windows, Linux, Unix para los clientes. La automatización del proceso de Gestión y control del avance silábico permitió que la Universidad Privada de Tacna trabajara de forma más eficiente y rápida ya que antes del sistema para realizar un informe de control del avance silábico, se demoraba o la información no era congruente, con el sistema el tiempo de control y gestión fue más eficaz y rápido.

El trabajo realizado por Huayamares L. (2012) titulado: implementación de un sistema de información para el área de registro civil, en la municipalidad distrital de pueblo nuevo, fue presentado en la Universidad Privada “Ada A. Byron” S.A.C para optar al título de Ingeniero de Sistemas, tenía como objetivo realizar un análisis, diseño e implementación de un sistema de información que modele el proceso de los registros civiles provinciales. Con esta investigación se pretendió mejorar la calidad de atención en la municipalidad para que los usuarios obtengan un mejor servicio en el área de registro civil haciéndolos más rápidos y eficientes. Para el desarrollo del proyecto se utilizó la metodología Proceso Unificado de Rational, para el diseño de la base de datos se usó SQL Server, la programación se realizó con el lenguaje de programación C Sharp, Rational Rose para el modelado del sistema.

El trabajo realizado por Pinargote M., y Sánchez C. (2014) titulado: sistema informático de gestión de ventas y servicios técnicos en la empresa s-compu del cantón pedernales, fue presentado en la Escuela Superior Politécnica Agropecuaria de Manabí Manuel Félix López, para optar al título de Ingeniero en Informática, cuyo objetivo fue desarrollar un sistema informático de control de gestión en la Empresa S-Compu del cantón Pedernales para optimizar los registros de ventas y servicios técnicos. Para el desarrollo del sistema se empleó la metodología de Prototipado, conjuntamente con los métodos inductivo-deductivo, el lenguaje de programación utilizado fue Visual Basic 2010, mediante el entorno de desarrollo integrado de Microsoft Visual Studio en su versión .NET y conjuntamente con la programación orientada a objetos, como gestor de base de datos se utilizó SQL Server 2008R2, también se usó Photoshop y Telerik. El Sistema mejoró los procesos de ventas y soporte técnico, brindando mayor rapidez y seguridad al ingreso y registro de datos.

El trabajo de Grado realizado por Malucín M. (2015) nombrado: sistema informático para mejorar la gestión comercial del almacén “Dangelo”, fue presentado en la Universidad Regional Autónoma de Los Andes, para obtener el título de Ingeniería en Sistemas e Informática, cuyo objetivo principal fue diseñar, estructurar e implementar un sistema informático mediante el cual se pueda mejorar la gestión comercial del almacén “D’ANGELO”. Para el desarrollo del Sistema se utilizó la metodología Cuantitativa y en la parte práctica se utilizaron las tecnologías de Visual Studio 2008 (ASP.NET), SQL Server 2008, entre otras que dieron solución al problema. Con la implementación del Sistema se facilitó el registro y manipulación de la información de los clientes, proveedores, productos, empleados de manera dinámica y automatizada.

López M. (2013) desarrolló un proyecto titulado: análisis, diseño y desarrollo de un sistema de información para soportar el proceso de inventario y préstamos de libros en la biblioteca de la Institución Educativa Alejandro Vélez Barrientos del municipio de Envigado, Antioquia utilizando la plataforma Visual Studio.Net 2010 y SQL Server, el cual fue presentado en la Universidad Nacional Abierta y a Distancia – UNAD de Medellín para optar al título de Tecnóloga en Sistemas. El objetivo de este trabajo fue analizar, diseñar y desarrollar un sistema de información para soportar el proceso de inventario y de préstamos de libros de la biblioteca de la Institución Educativa Alejandro Vélez Barrientos del Municipio de Envigado, Antioquia. La metodología utilizada en este proyecto es el ciclo de vida clásico o en cascada, junto con las herramientas tecnologías: Windows 7 Starter, Microsoft Visual Studio 2010: con sus herramientas y lenguaje de programación (Visual Studio. NET como Herramienta y Visual Basic Como Lenguaje de Programación), SQL Server 200 para la gestión de bases de datos, Planner para la gestión de cronogramas, Microsoft Office 2007 y Adobe Photoshop cs4. Con esto se logró mejorar el manejo del inventario, así como también otras actividades que son necesarias en la biblioteca.

2.4 Implementación de .net en los sistemas de información

Romero R. (2012) realizó un proyecto titulado: análisis, diseño e implementación de un sistema de información aplicado a la gestión educativa en centros de educación especial, este proyecto fue presentado en la Pontificia Universidad Católica del Perú para optar al título de Ingeniero en Informática y tenía como objetivo analizar, diseñar e implementar un sistema de información web orientado a la gestión educativa de un centro de educación especial. El propósito de esta plataforma fue posibilitar la administración y atención de los planes curriculares funcionales y terapéuticos para personas con necesidades especiales, así como consolidar el conocimiento de trastornos y promover la participación y evaluación continua entre padres y

especialistas. Para el desarrollo del proyecto se usó la metodología Agile Unified Process (AUP) y para la opción de la arquitectura se evaluaron varias como MVC, MVP y N-capas resultando una estructura de cuatro capas con funciones específica e independiente.

La implementación fue llevada a cabo mediante el IDE Microsoft Visual Web Developer 2010 Express y el lenguaje de programación C# soportado bajo .NET Framework 4.0. Para la construcción de las páginas se trabajó con ASP.NET Webforms y controles dinámicos de la librería Ajax Control Toolkit. La capa de Acceso a Datos fue construida bajo la tecnología Microsoft ADO.NET Entity Framework y en conexión con una base de datos PostgreSQL.

El trabajo de Grado realizado por Ramírez C. y Vélez G. (2015) titulado: automatización del registro y control de los procesos de hospedaje, restaurante y eventos del hotel-laboratorio "el higuerón" de la ESPAM MFL, fue presentado en la Escuela Superior Politécnica Agropecuaria de Manabí Manuel Félix López para optar al título de Ingeniero Informático, cuyo objetivo fue desarrollar un sistema informático web para automatizar el registro y control de los procesos de hospedaje, restaurante y eventos en el área de administración turística del Hotel-Laboratorio "El Higuerón". Para su desarrollo se utilizó Microsoft SQL Server para la base de datos y para la programación empleó el IDE Microsoft Visual Studio y sus herramientas como .NET, C#, ASP.NET. Además, se usó Bootstrap, JavaScript, Css, JQuery, AJAX, entre otras. Se obtuvo un sistema que permite el registro y control en los procesos del Hotel-Laboratorio y con ello se puede acceder a reportes del sistema de manera confiable y segura.

Pazmiño C. (2013) realizó un proyecto denominado: análisis, diseño, y desarrollo de un sistema de evaluación de desempeño laboral basado en competencias, para optar el título de Ingeniero Informático, el cual tenía como objetivo analizar, diseñar, y desarrollar un sistema informático que permita evaluar en forma eficiente y efectiva el desempeño de los empleados, en

relación a los estándares preestablecidos de la organización. Para ello se optó por utilizar la Metodología Ágil MSF (Microsoft Solution Framework), así como también se usó .NET Framework, ASP.NET para la creación de la capa de negocio, SQL Server como gestor de base de datos, Internet Information Services como servidor web, HTML, JavaScript, entre otros.

El Sistema web, mejoró los procesos de evaluación de desempeño laboral dentro de la empresa, la optimización de información tiene como efecto mejoras en el área de RRHH, ya que ahora los informes respectivos se pueden generar en forma automática lo que es un beneficio para la toma de decisiones.

El trabajo de Grado realizado por Cepeda M. (2014) titulado: implementación de un sistema informático para la gestión académica de la Unidad Educativa Intercultural Bilingüe "Corazón de la Patria", fue presentado en la Universidad Regional Autónoma de los Andes para optar al título de Ingeniero en Sistemas e Informática. Este trabajo tenía como objetivo implementar un sistema informático para mejorar la gestión académica de la Unidad Educativa Intercultural Bilingüe "Corazón de la Patria". Para el desarrollo se aplicó la metodología RUP (Rational Unified Process), se trabajó en Visual Studio utilizando el lenguaje de programación .NET, SQL Server para la base de datos y otras tecnologías óptimas para el desarrollo.

Mero S. (2014) desarrolló un proyecto denominado: sistema informático de control de ejecución de convenios y becas de la Unidad de Cooperación Interinstitucional de la Universidad Técnica Estatal de Quevedo. Este proyecto fue realizado para optar al título de Ingeniero en Sistemas y tenía como objetivo desarrollar un Sistema Informático de Control de Ejecución de Convenios y Becas para mejorar los procesos de registro y seguimiento de información en la

Unidad de Cooperación Interinstitucional. La aplicación ha sido desarrollada usando las siguientes herramientas: Microsoft Visual Studio.Net 2010, SQL Server 2005 para la base de datos, entre otras que también fueron indispensables. Con la implementación del sistema el registro de la información tiene un mayor aporte en la asignación de coordinadores para el mejoramiento del control y ejecución de convenio, facilitó el manejo de los avances de cumplimientos de convenios y académicos de parte de las becas, de igual manera facilitó el proceso de informes de cumplimiento de convenios y becas para la elaboración de las estadísticas de cumplimientos.

III. Marco teórico

En este apartado se darán a conocer una variedad de conceptos básicos indispensables para la compresión del ámbito del proyecto. Primeramente, se hará mención del significado de Sistema de Información, su evolución, así como también definición de la Arquitectura MVC y sus componentes. Además, se darán a conocer las tecnologías que se usaron para la programación.

3.1 Definición de sistema de información

Andreu, Ricart y Valor. (1991), definen a un Sistema de Información como un conjunto formal de procesos que, operan sobre una colección de datos estructurada de acuerdo a las necesidades de la empresa, recopila, elabora y distribuye selectivamente la información necesaria para la operación de dicha empresa y para las actividades de la dirección y control correspondientes, apoyando, al menos en parte, los procesos de toma de decisiones necesarios para desempeñar funciones de negocio de la empresa de acuerdo con su estrategia.

K. y J. Laudon. (1996), definen a un Sistema de Información como un conjunto de componentes interrelacionados que capturan, almacenan, procesan y distribuyen la información para apoyar la toma de decisiones, el control, análisis y visión de una organización.

3.1.1 Elementos que constituyen un sistema de información

- El hardware o equipo computacional, necesario para el funcionamiento del sistema. Está constituido por computadoras y otros dispositivos que pueden conectarse a ellas.
- El recurso humano que interactúa con el sistema de información, el cual está conformado por las personas que utilizan el sistema, alimentándolo con datos o utilizando los resultados que se generan.

- Los datos o información que son introducidos en el sistema, es decir las entradas que el sistema necesita para generar la información que se desea.
- Los programas (software) que son ejecutados por la computadora y producen diferentes resultados. Los cuales harán que los datos introducidos sean procesados correctamente y generen los resultados que se necesitan
- Políticas y reglas de operación, tanto en la parte funcional del proceso de negocio, así como también los mecanismos para hacer funcionar o trabajar una aplicación en la computadora.
- Las telecomunicaciones para la transmisión de texto, datos, imágenes y voz en forma electrónica.

3.1.2 Evolución de los sistemas de información

El origen de los Sistemas de Información se puede rastrear tan atrás como los censos (en donde se recopila, almacena, procesa y recupera información que posteriormente se usa para la toma de decisiones) que realizaban los babilonios y egipcios 4000 años a.C. Los sistemas de información surgen de la necesidad de organizar y administrar recurso y por lo tanto son tan antiguos como la civilización misma. Los egipcios y babilonios hacían censos, sin duda alguna, un censo es un sistema de información dado que se recolecta información, se procesa, y luego se provee la misma a alguien que la utiliza para la toma de decisiones.

Actualmente, se piensa en Sistemas de Información con sustento en las Tics. Existen diferentes tipos de sistemas de información (ej.: transaccionales, de apoyo a las decisiones, etc.) por lo que constituyen una familia de sistemas con diferentes características.

3.1.3 Clasificación de sistemas de información

Sistema de Información modelo pirámide

La primera clasificación se basa en la jerarquía de una organización y se llamó el modelo de la pirámide. Según la función a la que vayan destinados o el tipo de usuario final del mismo, los Sistemas de Información se pueden clasificarse en:

- Sistema de procesamiento de transacciones (TPS): Gestiona la información referente a las transacciones producidas en una empresa u organización, también se le conoce como Sistema de Información operativa.
- Sistemas de información gerencial (MIS): Orientados a solucionar problemas empresariales en general.
- Sistemas de soporte a decisiones (DSS): Herramienta para realizar el análisis de las diferentes variables de negocio con la finalidad de apoyar el proceso de toma de decisiones.
- Sistemas de información ejecutiva (EIS): Herramienta orientada a usuarios de nivel gerencial, que permite monitorizar el estado de las variables de un área o unidad de la empresa a partir de información interna y externa a la misma. Estos sistemas de información no surgieron simultáneamente en el mercado; los primeros en aparecer fueron los TPS, en la década de los 60, sin embargo, con el tiempo, otros sistemas de información comenzaron a evolucionar. Los primeros proporcionan información a los siguientes a medida que aumenta la escala organizacional
- Sistemas de automatización de oficinas (OAS): Aplicaciones destinadas a ayudar al trabajo diario del administrativo de una empresa u organización.
- Sistema Planificación de Recursos (ERP): Integran la información y los procesos de una organización en un solo sistema.

- Sistema experto (SE): Emulan el comportamiento de un experto en un dominio concreto.

Sistemas de información estratégicos

Su función principal es crear una diferencia con respecto a los competidores de la organización que hagan más atractiva a ésta para los potenciales clientes. Por ejemplo, en la banca, hace años que se implantaron los cajeros automáticos, pero en su día, las entidades que primero ofrecieron este servicio disponían de una ventaja con respecto a sus competidores, y hoy día cualquier entidad que pretenda ofrecer servicios bancarios necesita contar con cajeros automáticos si no quiere partir con una desventaja con respecto al resto de entidades de este sector. En este sentido, los cajeros automáticos se pueden considerar sistemas de información estratégicos.

Su función es lograr ventajas que los competidores no posean, tales como ventajas en costos y servicios diferenciados con clientes y proveedores. Apoyan el proceso de innovación de productos dentro de la empresa. Suelen desarrollarse dentro de la organización, por lo tanto, no pueden adaptarse fácilmente a paquetes disponibles en el mercado. Entre las características más destacables de estos sistemas se pueden señalar:

- Cambian significativamente el desempeño de un negocio al medirse por uno o más indicadores clave, entre ellos, la magnitud del impacto.
- Contribuyen al logro de una meta estratégica.
- Generan cambios fundamentales en la forma de dirigir una compañía, la forma en que compite o en la que interactúa con clientes y proveedores.
- Si los recursos tecnológicos están heterogéneamente distribuidos a lo largo de la competencia y si a las compañías que carecen de éstos les es más costoso desarrollarlos, adquirirlos y usarlos para implementar una estrategia en comparación con las empresas que

ya los han usado para implementar esa misma estrategia, estos recursos pueden ser utilizados como fuente de ventaja competitiva sostenida.

Según el entorno de aplicación

- Entorno decisional: este es el entorno en el que tiene lugar la toma de decisiones; en una empresa, las decisiones se toman a todos los niveles y en todas las áreas (otra cosa es si esas decisiones son estructuradas o no), por lo que todos los SI de la organización deben estar preparados para asistir en esta tarea, aunque típicamente, son los DSS los que se encargan de esta función. Si el único sistema de información de una compañía preparado para ayudar a la toma de decisiones es el DSS, éste debe estar adaptado a todos los niveles jerárquicos de la empresa.
- Entorno transaccional: una transacción es un suceso o evento que crea/modifica los datos. El procesamiento de transacciones consiste en captar, manipular y almacenar los datos, y también, en la preparación de documentos; en el entorno transaccional, por tanto, lo importante es qué datos se modifican y cómo, una vez que ha terminado la transacción. Los TPS son los sistemas de información típicos que se pueden encontrar en este entorno.

Sistemas de información de espionaje

La mayor parte de sistemas de información operan con el conocimiento de los agentes sobre los que se recaba información, el auge de las comunicaciones electrónicas ha hecho que proliferen sistemas secretos de espionaje como por ejemplo el programa PRISM por el cual la Agencia de Seguridad Nacional (NSA) instituida por el gobierno estadounidense ha operado desde 2007, espiando a líderes y presidentes de otros países (aliados y adversarios de Estados Unidos), y se ha afirmado que tiene capacidad para interceptar decenas de miles de comunicaciones telefónicas por

minuto. Gran parte de lo que se conoce sobre dicho sistema de información se conoció a partir del escándalo por las filtraciones de Edward Snowden (2013).

3.2 La importancia de los sistemas de información en las empresas

Las empresas u organizaciones siempre han utilizaron sistemas que les permiten administrar el manejo de su información, años atrás no era necesaria una computadora para reconocer la existencia de un SI dentro de una empresa. Sin embargo, es cada vez más necesario el disponer de sistemas de información basados en computadoras por los beneficios que estos proporcionan: reducción de errores provocados por las personas a través del control de las entradas, velocidad en el procesamiento de datos, posibilidad de realizar tediosos análisis sobre los mismos, reducción de espacio físico destinado a su almacenamiento, agilidad al momento de buscar algún dato en particular, y otros tipos de ventajas que podrían lograrse en caso de enfocarse en el uso estratégico de los mismos. En los entornos de negocio actuales, disponer de una buena gestión en el uso de los sistemas de información se convierte en una estrategia que pueden utilizar las empresas para hacer frente a sus fuerzas competitivas.

Es importante tener una comprensión básica de los sistemas de información para entender cualquier otra área funcional en la empresa, por eso es importante también, tener una cultura informática en nuestras organizaciones que permitan y den las condiciones necesarias para que los sistemas de información logren los objetivos citados anteriormente. Muchas veces las organizaciones no han entrado la etapa de cambio hacia la era de la información sin saber que es un riesgo muy grande de fracaso debido a las amenazas del mercado y su incapacidad de competir, por ejemplo, las TI que se basan en Internet se están convirtiendo rápidamente en un ingrediente necesario para el éxito empresarial en el entorno global y dinámico de hoy.

Por lo tanto, la administración apropiada de los sistemas de información es un desafío importante para los gerentes. Así la función de los SI representa:

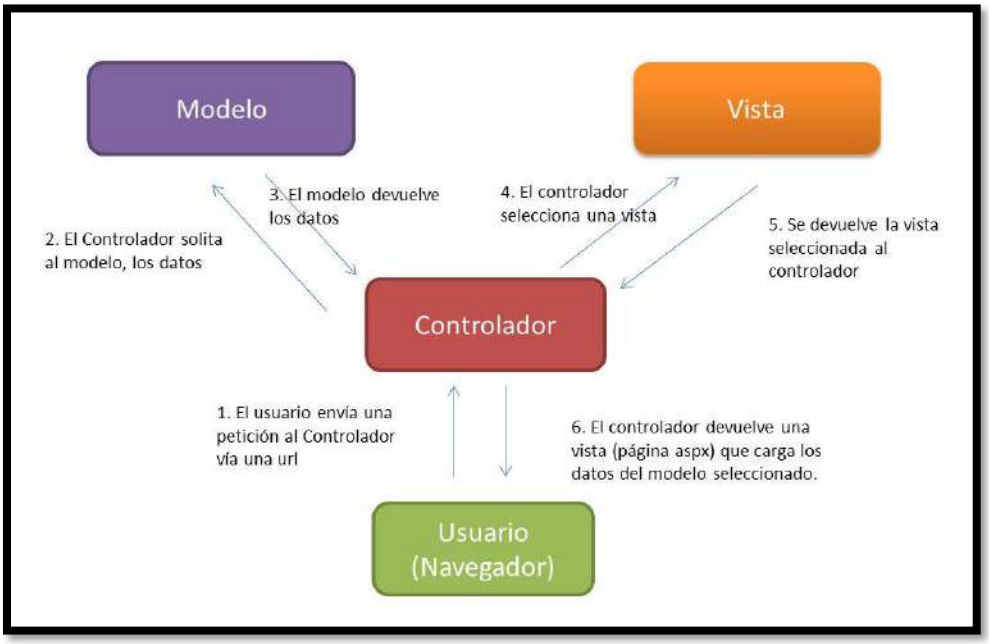
- Un área funcional principal dentro de la empresa, que es tan importante para el éxito empresarial como las funciones de contabilidad, finanzas, administración de operaciones, marketing, y administración de recursos humanos.
- Una colaboración importante para le eficiencia operacional, la productividad y la moral del empleado, y el servicio y satisfacción del cliente.
- Una fuente importante de información y respaldo importante para la toma de decisiones efectivas por parte de los gerentes.
- Un ingrediente importante para el desarrollo de productos y servicios competitivos que den a las organizaciones una ventaja estratégica en el mercado global.
- Una oportunidad profesional esencial, dinámica y retadora para la humanidad.

3.3 Arquitectura MVC

La arquitectura MVC (Model, View, Controller) fue introducida como parte de la versión Smalltalk-80 del lenguaje de programación Smalltalk. Es un patrón de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones.

MVC propone la construcción de tres componentes que son el modelo, la vista y el controlador, es decir, por un lado, define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón de arquitectura de software se basa en las ideas de reutilización de código y la separación de conceptos, estas características buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

Ilustración 1 Arquitectura MVC



Descripción de los componentes de la Arquitectura MVC.

Componentes

- El Modelo: Es la representación de la información con la cual el sistema opera, por lo tanto, gestiona todos los accesos a dicha información, tantas consultas como actualizaciones. Envía a la vista aquella parte de la información que en cada momento se le solicita para que sea mostrada. Las peticiones de acceso o manipulación de información llegan al modelo a través del controlador.

  El modelo es el responsable de:

  - Acceder a la capa de almacenamiento de datos.
  - Define las reglas de negocio (la funcionalidad del sistema).
  - Lleva un registro de las vistas y controladores del sistema.

- El Controlador: Responde a eventos e invoca peticiones al modelo cuando se hace alguna solicitud sobre la información. Puede enviar comandos a su vista asociada si se solicita un cambio en la forma en que se presenta el modelo, se podría decir que el controlador hace de intermediario entre la vista y el modelo.

  El controlador es responsable de:

  - Recibir los eventos de entrada.
  - Contiene reglas de gestión de eventos.

- La Vista: Presenta el modelo en un formato adecuado para interactuar, por tanto, requiere de del modelo la información que debe representar como salida.

  Las vistas son responsables de:

  - Recibir datos del modelo y mostrarlos al usuario.
  - Tienen un registro de su controlador asociado
  - Pueden dar el servicio de "Actualización", para que sea invocado por el controlador o por el modelo.

3.3.1 Uso de MVC en las aplicaciones web

Originalmente la Arquitectura MVC fue desarrollada para aplicaciones de escritorio, pero en la actualidad ha sido ampliamente utilizada como arquitectura para el diseño e implementación de aplicaciones web usando los principales lenguajes de programación. Se han desarrollado multitud de frameworks, comerciales y no comerciales, que implementan este patrón.

Los primeros frameworks MVC para desarrollo web plantean un enfoque de cliente ligero en el que casi todas las funciones, tanto de la vista, el modelo y el controlador recaen en el servidor. En este enfoque, el cliente manda la petición de cualquier hiperenlace o formulario al controlador y después recibe de la vista una página completa y actualizada, tanto el modelo como el controlador

están completamente alojados en el servidor. En la actualidad existen frameworks como JavaScriptMVC, Backbone o jQuery que permiten que ciertos componentes MVC se ejecuten parcial o totalmente en el cliente.

3.4 Entorno de desarrollo integrado (IDE)

Un entorno de desarrollo integrado (IDE en inglés de Integrated Development Environment), es un programa informático compuesto por un conjunto de herramientas de programación. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI).

3.5 Visual Studio

Microsoft Visual Studio es un IDE (Entorno de desarrollo integrado) para sistemas operativos Windows. Visual Studio punto Net soporta varios lenguajes de programación como lo son C#, C++, ASP.NET, Ruby, Java, Python, PHP y Visual Basic .NET. Visual Studio permite a los desarrolladores crear sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Así, se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos y consolas, entre otros.

Visual Studio ha sido diseñado para asegurar que los desarrolladores puedan proporcionar un flujo continuo de valor para sus negocios. La interfaz ha sido mejorada para eliminar el desorden significativo de la pantalla y proporciona un rápido acceso a las funciones más utilizadas.

3.6 Microsoft .net

.NET es una arquitectura de tecnología, desarrollada por Microsoft para la creación y distribución del software como un servicio. Mediante las herramientas que proporciona esta tecnología se pueden crear aplicaciones basadas en servicios para la web. Este framework de Microsoft que hace un énfasis en la transparencia de redes, con independencia de plataforma de hardware y que permita un rápido desarrollo de aplicaciones. Provee un extenso conjunto de soluciones predefinidas para necesidades generales de la programación de aplicaciones, y administra la ejecución de los programas escritos específicamente con la plataforma.

Proporciona las herramientas y servicios que se necesitarán en el desarrollo. Permite el desarrollo de aplicaciones a través del uso de un conjunto de herramientas y servicios que proporciona, estas pueden agruparse en tres bloques principales: Entorno de Ejecución Común, la jerarquía de clases básica de la plataforma (.NET Framework Base Classes) y el motor de generación de interfaz de usuario, que permite crear interfaces para la web o para entornos de Windows.

3.7 Asp.net

ASP.NET es un modelo de desarrollo Web unificado que incluye los servicios necesarios para crear aplicaciones Web empresariales con el código mínimo. ASP.NET forma parte de .NET Framework y al codificar las aplicaciones ASP.NET tiene acceso a las clases en .NET Framework. El código de las aplicaciones puede escribirse en cualquier lenguaje compatible con el Common Language Runtime (CLR), entre ellos Microsoft Visual Basic, C#, JScript .NET y J#. Estos lenguajes permiten desarrollar aplicaciones ASP.NET que se benefician del Common Language Runtime, seguridad de tipos, herencia, etc.

Componentes de ASP.NET:

- Marco de trabajo de página y controles
- Compilador de ASP.NET
- Infraestructura de seguridad
- Funciones de administración de estado
- Configuración de la aplicación
- Supervisión de estado y características de rendimiento
- Capacidad de depuración
- Marco de trabajo de servicios Web XML
- Entorno de host extensible y administración del ciclo de vida de las aplicaciones
- Entorno de diseñador extensible

3.8 Asp.net Core

ASP.NET Core es un framework multiplataforma, de alto rendimiento y de código abierto para la creación de aplicaciones modernas, basadas en la nube y conectadas a Internet. Inicialmente considerado como ASP.NET vNext, el marco se iba a llamar ASP.NET 5, pero, para evitar implicar que es una actualización de la estructura existente de ASP.NET, Microsoft cambió el nombre a ASP.NET Core en la versión 1.0.

Características:

- Experiencia de desarrollador sin compilación (es decir, la compilación es continua, por lo que el desarrollador no tiene que invocar el comando de compilación).
- Marco modular distribuido como paquetes NuGet.
- Tiempo de ejecución optimizado para la nube (optimizado para Internet).

- Host-agnostic a través de Open Web Interface para .NET (OWIN) de apoyo, se ejecuta en IIS o independiente.
- Una historia unificada para la creación de la interfaz web y las API web.
- Un sistema de configuración basado en la nube.
- Una tubería de petición HTTP ligera y modular.
- Creación y ejecución de aplicaciones multiplataforma ASP.NET Core en Windows, Mac y Linux.

3.9 Entity framework

Entity Framework es un conjunto de tecnologías en ADO.NET que soportan el desarrollo de aplicaciones de software orientadas a datos. Entity Framework permite a los desarrolladores trabajar con datos en forma de objetos y propiedades específicos de dominio, como clientes y direcciones de clientes, sin tener que preocuparse por las tablas y columnas de base de datos subyacentes en las que se almacenan estos datos. Con esta herramienta los desarrolladores pueden trabajar en un nivel superior de abstracción cuando tratan con datos y pueden crear y mantener aplicaciones orientadas a datos con menos código que en aplicaciones tradicionales.

3.10 Bootstrap

Bootstrap es un framework o conjunto de herramientas de Código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones, cuadros, menús de navegación y otros elementos de diseño basado en HTML y CSS, así como, extensiones de JavaScript opcionales adicionales. Tiene un soporte relativamente incompleto para HTML5 y CSS 3, pero es compatible con la mayoría de los navegadores web. La información básica de compatibilidad de sitios web o aplicaciones está disponible para todos los dispositivos y navegadores. Existe un concepto de compatibilidad parcial que hace disponible la información

básica de un sitio web para todos los dispositivos y navegadores. Desde la versión 2.0 soporta diseños sensibles, es decir que el diseño gráfico de la página se ajusta dinámicamente, tomando en cuenta las características del dispositivo usado (Computadoras, tabletas, teléfonos móviles).

3.11 JavaScript

JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico. Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. Sin embargo, existe una forma de JavaScript del lado del servidor (Server-side JavaScript o SSJS), que se usa en aplicaciones externas a la web, por ejemplo, en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets).

Características

- Imperativo y estructurado: es compatible con gran parte de la estructura de programación de C (por ejemplo, sentencias if, bucles for, sentencias switch, etc.). Con una salvedad, en parte: en C, el ámbito de las variables alcanza al bloque en el cual fueron definidas; sin embargo, JavaScript no es compatible con esto, puesto que el ámbito de las variables es el de la función en la cual fueron declaradas. Esto cambia con la versión de ECMAScript 2015, ya que añade compatibilidad con block scoping por medio de la palabra clave let. Como en C, JavaScript hace distinción entre expresiones y sentencias. Una diferencia sintáctica con respecto a C es la inserción automática de punto y coma, es decir, en JavaScript los puntos y coma que finalizan una sentencia pueden ser omitidos.

- Dinámico:

- Tipado dinámico: como en la mayoría de lenguajes de scripting, el tipo está asociado al valor, no a la variable. Por ejemplo, una variable x en un momento dado puede estar ligada a un número y más adelante, religada a una cadena. JavaScript es compatible con varias formas de comprobar el tipo de un objeto, incluyendo duck typing. Una forma de saberlo es por medio de la palabra clave typeof.
- Objetual: está formado casi en su totalidad por objetos. Los objetos son arrays asociativos, mejorados con la inclusión de prototipos. Los nombres de las propiedades de los objetos son claves de tipo cadena: obj.x = 10 y obj['x'] = 10 son equivalentes. Las propiedades y sus valores pueden ser creados, cambiados o eliminados en tiempo de ejecución. La mayoría de propiedades de un objeto (y aquellas que son incluidas por la cadena de la herencia prototípica) pueden ser enumeradas a por medio de la instrucción de bucle for... in.
- Evaluación en tiempo de ejecución: incluye la función eval que permite evaluar expresiones expresadas como cadenas en tiempo de ejecución. Por ello se recomienda que eval sea utilizado con precaución y que se opte por utilizar la función JSON.parse() en la medida de lo posible, pues resulta mucho más segura.

- Funcional: las funciones se les suele llamar ciudadanos de primera clase; son objetos en sí mismos. Como tal, poseen propiedades y métodos, como: .call() y .bind(). Una función anidada es una función definida dentro de otra. Esta es creada cada vez que la función externa es invocada. Además, cada función creada forma una clausura; es el resultado de evaluar un ámbito conteniendo en una o más variables dependientes de otro ámbito externo, incluyendo constantes, variables locales y argumentos de la función externa llamante. El

resultado de la evaluación de dicha clausura forma parte del estado interno de cada objeto función, incluso después de que la función exterior concluya su evaluación.

- Prototípico
  - Prototipos: usa prototipos en vez de clases para el uso de herencia. Es posible llegar a emular muchas de las características que proporcionan las clases en lenguajes orientados a objetos tradicionales por medio de prototipos en JavaScript.
  - Funciones como constructores de objetos: las funciones también se comportan como constructores. Prefijar una llamada a la función con la palabra clave new crear una nueva instancia de un prototipo, que heredan propiedades y métodos del constructor (incluidas las propiedades del prototipo de Object). ECMAScript 5 ofrece el método Object.create, permitiendo la creación explícita de una instancia sin tener que heredar automáticamente del prototipo de Object. La propiedad prototype del constructor determina el objeto usado para el prototipo interno de los nuevos objetos creados. Se pueden añadir nuevos métodos modificando el prototipo del objeto usado como constructor.

3.12 Hojas de estilo en cascada (css)

Es una tecnología que permite crear páginas web de una manera más exacta. Gracias a las CSS se pueden incluir en las páginas web márgenes, tipos de letra, fondos, colores, etc. Las Hojas de Estilo en Cascada se escriben dentro del código HTML de la página web, solo en casos avanzados se pueden escribir en un archivo a parte y enlazar la página con ese archivo, pero igual se puede utilizar en un atributo en casi todas las etiquetas de HTML.

El lenguaje CSS permite presentar, de manera estructurada, un documento que fue escrito en un lenguaje de marcado. Se usa especialmente en el diseño visual de un sitio web cuando las

páginas se hallan escritas en XML o HTML. El CSS se desarrolló en distintos niveles. El CCS1 ya no se emplea, mientras que el CSS2 funciona como recomendación. El CSS3, que se divide en varios módulos, es el lenguaje que se está tomando como estándar. Lo que hace el CSS es encargarse de la descripción de las formas y de la sintaxis del lenguaje de marcado. De esta manera describe cómo se tienen que renderizar (generar las imágenes) los elementos que aparecen en pantalla.

3.13 Microsoft SQL Server

Microsoft SQL Server es un conjunto completo de tecnologías y herramientas específicas para la empresa que permiten que el personal obtenga de la información el máximo valor al menor costo de propiedad (MICROSOFT, 2013).

SQL Server es un sistema de manejo de bases de datos del modelo relacional, desarrollado por Microsoft. El lenguaje de desarrollo utilizado es Transact-SQL (TSQL) el cual puede ser utilizado por línea de comandos o mediante la interfaz gráfica de Management Studio, este es una implementación del estándar ANSI del lenguaje SQL, utilizado para manipular y recuperar datos (DML), crear tablas y definir relaciones entre ellas (DDL).

El motor de base de datos de SQL Server ofrece almacenamiento más seguro y confiable tanto para los datos relacionales como estructurados, lo que permite crear y administrar aplicaciones de datos altamente disponibles y con mayor rendimiento. Además, combina lo mejor en análisis, información, integración y notificación.

3.13.1 Características de Microsoft SQL Server

- Soporte de transacciones.
- Escalabilidad, estabilidad y seguridad.

- Soporta procedimientos almacenados.
- Incluye un potente entorno gráfico de administración, que permite el uso de comandos DDL y DML gráficamente.
- Permite trabajar en modo cliente-servidor, donde la información y datos se alojan en el servidor y las terminales o clientes de la red sólo acceden a la información.
- Permite administrar información de otros servidores de datos.

3.13.2 Herramientas de SQL Server

Aparte de sus capacidades elementales como herramienta de gestión de bases de datos relacionales (crear tablas, definir las relaciones, gestión de transacciones, crear índices etc.), posee otras herramientas las cuales se enlistan a continuación:

- Respaldos y recuperaciones: Existe una herramienta integrada en el SQL Server que posibilita un respaldo completo o diferencial, de acuerdo con el modelo de recuperación (Recovery Model) predefinido a la base de datos; y una recuperación completa o a un punto de tiempo. Aparte de un respaldo de la base de datos se puede respaldarlos a través de un guión (con o sin los datos). A partir de 2008, se puede comprimir los archivos de respaldo.
- Compresión: A partir de 2008 se añadió la opción de compresión que posibilita comprimir el tamaño físico de las tablas y los índices, y utilizar más eficientemente el volumen de los discos rígidos y reducir operaciones de IO (lo que aumenta la carga del CPU). Como ya ha sido mencionado, se puede comprimir también los archivos de respaldo.
- Replicación, alta disponibilidad, y recuperación de desastres: Algunas herramientas posibilitan crear réplicas parciales o completos de las bases de datos, mejorar la disponibilidad, y recuperar de desastres; aparte de la opción de respaldar y recuperar las bases de datos, una opción que se supone que es muy lenta; se mencionan a continuación:

  - La replicación: que posibilita crear una réplica sincronizada de la base de datos
  - El Mirroring: que ejecuta en el servidor de espejo cada actualización que se ejecuta en el servidor de origen
  - Log Shipping: que posibilitan almacenar una copia sincronizada a través de archivos de registro (Log) con todas las actualizaciones en el servidor de origen.
  - Herramientas de alta disponibilidad y recuperación de desastres (HADR, desde 2012) que solucionan problemas de Mirroring en relación con el tiempo de reacción de los fallos técnicos y la disponibilidad de las copias en tiempos de paz.

- El agente y la programación de tareas: El agente es el servicio encargado de la programación de tareas, y se encarga de ejecutarlas independientemente. Generalmente ejecuta tareas de mantenimiento, tareas complejas de ETL, respaldos, etc.
- Service Broker: Una tecnología que fue agregada en 2005, la cual implementa arquitectura orientada a servicios, y que posibilita ejecuciones asíncronas: primero para enviar mensajes entre distintas aplicaciones que se ejecutan simultáneamente, pero también para ejecutar procedimientos asíncronamente, en la manera de dispar y olvidar: un procedimiento que se ejecuta en una sesión diferente de la sesión que lo inicio, y ambos procedimientos siguen ejecutado independientemente uno del otro.
- Enviar Correos Electrónicos: SQL Server tiene la capacidad de enviar correos electrónicos a través del código. Esta capacidad se utiliza generalmente para enviar alertas sobre problemas en el sistema, pero también cuando el proceso termino exitosamente.
- Full Text Search (Búsqueda de Textos completos): Una herramienta que posibilita indexar columnas textuales como textos y no solo como cadenas; y ejecutar búsquedas complejas dependientes en el sentido del texto y en el idioma.

- Rastrear: Estas herramientas incluyen el Trace que posibilita rastrear actividades con el fin de mantener cargas y fallos, y seguridad de datos. El Profiler que posibilita rastrear los comandos que se ejecutan y los eventos que se ocurren en el servidor, y el Extended Events que fue agregado en 2008 y cambia el profiler gracias a su baja consumición de recursos y la influencia sobre el rendimiento del servidor.
- Herramientas de Inteligencia empresarial: Una instalación típica incluye también las herramientas de BI (Inteligencia empresarial):
    - SSIS (SQL Server Integration Services): Una herramienta de ETL que posibilita la extracción de datos de distintos orígenes (no solo SQL Server), la transformación de dichos datos, y la carga (generalmente pero no obligatoriamente a almacén de datos).
    - SSAS (SQL Server Analysis Services): Una herramienta para crear Bases de Datos Multidimensionales, que se puede explorar mediante extracciones de datos en distintos niveles de agrupación, profundización (Drill Down) de una suma a sus detalles, y utilización de MDX (un lenguaje parecido a SQL, adaptado a bases de datos multidimensionales).
    - SSRS (SQL Server Reporting Services): Una herramienta para crear y dar formato a informes, otorgar derechos de contemplación en ellos, y su distribución. Se puede contemplarlos con un Navegador web, y se puede exportarlos a archivos de Excel, PDF, etc. los datos se extraen generalmente del almacén de datos o del OLAP.

IV. Marco metodológico

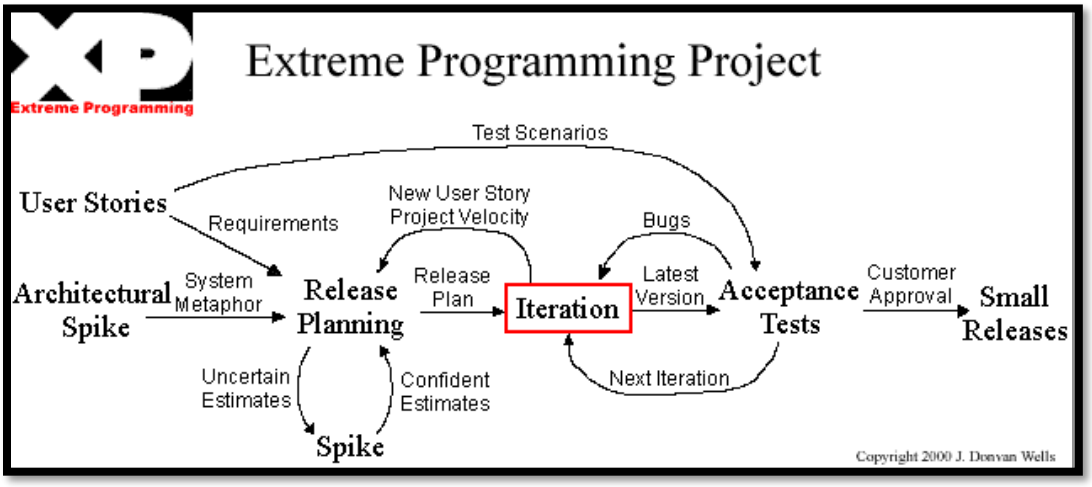
4.1 Metodología extreme programming

La metodología a utilizar en este proyecto es la metodología XP (Extreme Programming o Programación Extrema), es una de las llamadas metodologías ágiles de desarrollo de software, también es una de las metodologías más exitosas de estos tiempos. Propone una metodología de trabajo basada en la simplicidad y agilidad, permitiendo una interacción más cercana con el cliente y sobre todo entregar el software que los clientes necesitan en el momento en que lo necesitan según Joskowicz J. (2008). Este tipo de programación es la adecuada para los proyectos con requerimientos imprecisos y muy cambiantes; está diseñada para el desarrollo de aplicaciones que requieran un grupo de programadores pequeño, donde la comunicación sea más factible. La comunicación es un punto importante y debe realizarse entre los programadores, los jefes de proyecto y los clientes.

La metodología como se menciona en el libro “Scrum y XP desde las trincheras” enfatiza en el trabajo en equipo, tanto gerentes como clientes y desarrolladores son parte del mismo equipo dedicado a entregar un software de calidad (Kniberg, H. 2007).

El ciclo de vida de un proyecto con metodología XP se separa en fases o iteraciones, las cuales se muestran en la siguiente ilustración.

Ilustración 2 Fases de la Metodología XP



Esquematización de las fases de la metodología XP.

De acuerdo a la ilustración, el proyecto consta de las siguientes Fases para su realización.

4.1.1 Exploración del proyecto

La primera es la Fase de Exploración del Proyecto, en esta se determina el alcance general del proyecto, es donde se explora por primera vez el proyecto y el cliente define lo que necesita. Al mismo tiempo el equipo de desarrollo se familiariza con las herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Esta fase toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología.

4.1.2 Planificación

La segunda Fase es la de Planificación, en esta fase se acuerdan el orden en que deberán implementarse los módulos a desarrollar, las prioridades de cada funcionalidad, definidas por el cliente y el desarrollador y la presentación de los avances por parte del desarrollador hacia el cliente. La planificación se puede realizar basándose en el tiempo o el alcance. La velocidad del proyecto es utilizada para establecer cuántas historias de usuario se pueden implementar antes de

una fecha determinada o cuánto tiempo tomará implementar un conjunto de historias de usuario. El resultado de esta fase es un Plan de Entregas, o “Release Plan”.

4.1.3 Análisis y diseño

La tercera Fase es la de Análisis y Diseño, en donde se modela el sistema en general, se generan las clases necesarias y bases de datos para el desarrollo de sistema junto a una presentación por parte del desarrollador. El diseño XP sigue rigurosamente el principio MS (mantenlo sencillo). Un diseño sencillo siempre se prefiere sobre una representación más compleja. Además, el diseño guía la implementación de una historia conforme se escribe.

La metodología XP estimula el uso de las tarjetas CRC como un mecanismo eficaz para pensar en el software en un contexto orientado a objetos. Las tarjetas CRC (clase-responsabilidad-colaborador) identifican y organizan las clases orientadas a objetos que son relevantes para el incremento actual de software. Las tarjetas CRC son el único producto del trabajo de diseño que se genera como parte del proceso XP (Pressman, R. 2010).

4.1.4 Codificación

La cuarta Fase es la de Codificación. Para el desarrollador la codificación debe hacerse ateniéndose a estándares de codificación ya creados. Programar bajo estándares mantiene el código consistente y facilita su comprensión y escalabilidad. A la hora de codificar no se sigue la regla de XP que aconseja crear test de funcionamiento con entornos de desarrollo antes de programar. Los test se obtendrán de la especificación de requisitos ya que en ella se especifican las pruebas que deben pasar las distintas funcionalidades del programa, procurando codificar pensando en las pruebas que debe pasar cada funcionalidad.

Un concepto clave durante la codificación es la programación por parejas. La metodología XP recomienda que dos personas trabajen juntas en una estación de trabajo con el objeto de crear código para una historia. A medida que las parejas de programadores terminan su trabajo, el código que desarrollan se integra con el trabajo de los demás. Esta estrategia de "integración continua" ayuda a evitar los problemas de compatibilidad de interfaces y brinda un ambiente de "prueba de humo" que ayuda a descubrir a tiempo los errores (López P. y Francisco R. s/f).

4.1.5 Aceptación

La quinta Fase es la de Aceptación, es donde el cliente prueba el sistema para evaluar si cumple con los requerimientos especificados, de no ser así, se codifica nuevamente hasta obtener un producto óptimo de acuerdo a las condiciones del cliente.

4.1.6 Puesta en marcha

La última Fase es la de Puesta en Marcha, donde se implementa el sistema para su uso final y se hacen las capacitaciones necesarias al personal involucrado para el uso del nuevo sistema junto a una presentación formal de éste.

4.2 Recursos

4.2.1 Hardware

Para el desarrollo de este proyecto se utilizarán cuatro notebooks con las siguientes características:

Tabla 1 Hardware Utilizado

Hardware Utilizado	
Notebook 1	
Marca	Lenovo
Procesador	AMD A8-6410, 2.00 GHz
Sistema Operativo	Windows 10, 64 bits
Memoria RAM	12GB
Disco Duro	1 TB

Tabla 1: Hardware Utilizado

4.2.2 Software

Tabla 2 Software Utilizado

Software Utilizado	
Herramienta	Uso
Windows 10, 64 bits	Sistema Operativo con el que se trabajó
Visual Studio 2017, 64 bits	Plataforma base para desarrollar el sistema
HTML 5, CSS3, Bootstrap 4, ASP .NET, MVC 3	Lenguajes de programación en los cuales se desarrolló el sistema, todos ellos compatibles con las plataformas de desarrollo de Microsoft.
Ajax, JQuery, JavaScript	Recursos utilizados para lograr la interactividad de la interfaz gráfica del sistema con el usuario.
SQL Server 2017	Motor de Base de Datos usado para el almacenamiento de la información necesaria.

Tabla 2: Software Utilizado

V. Desarrollo

5.1 Fase de exploración del proyecto

5.1.1 Definición del sistema

El presente proyecto de titulación tiene como misión la construcción de un Sistema de Información web para una empresa inmobiliaria, con la finalidad de controlar y gestionar los procesos de las áreas involucradas, dando solución a los problemas detectados por el cliente y usuarios finales. Además, se tendrá un control de acceso hacia la plataforma y la notificación vía correo electrónico hacia el personal autorizado, mismo que no existe en el sistema actual.

La solución propuesta con anterioridad se distribuye de acuerdo a los siguientes módulos:

- El módulo de Administración de Proveedores, presenta toda la información de los Proveedores, Cuentas Bancarias, Contactos, que se ingresarán al sistema, tanto los datos personales e información correspondiente.
- El módulo de Administración de Órdenes de Compras y Estimaciones, el cual contiene información relacionada con las compras realizadas, en estas se incluye datos de Proveedores, Tipos de Compra, Comprador, Fechas, Facturas, entre otros.

5.1.2 Requerimientos

Este apartado tiene como objetivo establecer lo que el sistema debe hacer, es decir especificar lo requisitos, esto incluye definir los límites del sistema, una interfaz de usuario, realizar una estimación del costo y tiempo de desarrollo. Con esto se pretende que los desarrolladores entiendan las necesidades del cliente y tengan una base para estimar recursos y el tiempo de desarrollo del sistema. Los requerimientos sean divididos en dos grupos: Los

funcionales, que describen las funciones que el software va a ejecutar; y los no funcionales, que especifican los criterios que pueden usarse para juzgar la operación de un sistema.

5.1.2.1 Requerimientos funcionales

Los requerimientos funcionales ofrecen una descripción detallada del comportamiento y de las necesidades del sistema, así como soluciones a posibles situaciones adversas o anómalas tales como datos inválidos, errores, fallos del sistema entre otras. Por lo tanto, para poder realizar correctamente el proyecto en cuestión debemos asegurarnos que tenemos claros todos los requerimientos a cumplir. En el siguiente listado se muestran los requisitos mínimos debe cumplir el sistema.

- RF01. El sistema permitirá el acceso solo al personal autorizado que cuente con un usuario y contraseña
- RF02. El sistema debe mostrar una lista de todos los Proveedores dados de alta.
- RF03. El sistema debe permitir la creación, edición, consulta y eliminación de los Proveedores.
- RF04. El sistema debe permitir, la creación, edición, consulta y eliminación de las Cuentas Bancarias, Contactos y Documentos, los cuales deben de pertenecer a un Proveedor.
- RF05. El sistema debe mostrar una lista de todas las Órdenes de Compra y Estimaciones dadas de alta.
- RF06. El sistema debe permitir la creación, edición, consulta y eliminación de las Órdenes de Compra y Estimaciones, considerando que poder crear una Orden de Compra o Estimación primeramente debe de existir Proveedores dados de alta.

5.1.2.2 Requerimientos no funcionales

Los requerimientos no funcionales son con características que de una u otra forma puedan limitar el sistema. Estos requerimientos se basan en restricciones impuestas por el cliente y usuarios o bien surgidas por sucesos imprevistos y que afectan al diseño final. Normalmente son cuantificables. Algunos ejemplos son, el rendimiento (en tiempo y espacio), interfaces de usuario, fiabilidad, mantenimiento, seguridad, portabilidad, estándares, etc. En el siguiente listado se muestran los requerimientos no funcionales que debe cumplir el sistema.

- RNF01. El sistema deberá estar alojado en un servidor con Internet para que los usuarios que se conecten puedan usarlo sin problemas.
- RNF02. Se deberá contar con un servidor de Base de Datos para el alojamiento y acceso de toda la información.
- RNF03. El Sistema debe estar disponible cuando el usuario desee acceder al sistema.
- RNF04. El sistema deberá estar vinculado con otro Sistema (AX).

5.1.3 Creación de casos de uso

En este apartado se determina la descripción del Sistema en términos de una secuencia de acciones, lo cual permite saber con más claridad los requerimientos y las condiciones del cliente para el uso del sistema a desarrollar, el propósito de esto es facilitar el acuerdo entre los desarrolladores y los usuarios acerca de cómo debería funcionar el sistema.

5.1.3.1 Identificar y esquematizar diagramas de casos de uso

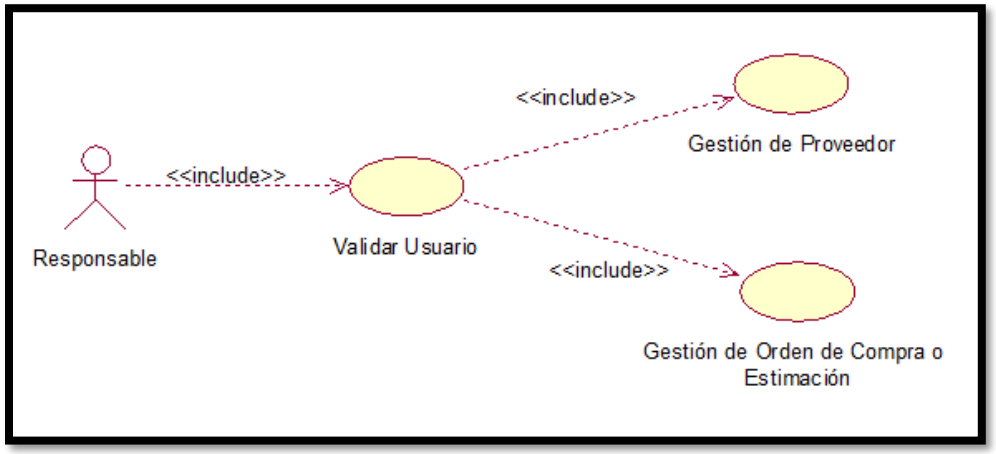
A partir de los requerimientos definidos y analizados con anterioridad, se estudiarán los casos de uso para definir de forma más adecuada los procesos y la interacción de los usuarios con el sistema.

Antes de describir que es un caso de uso primero se dará una breve definición de un actor. Un actor representa el rol genérico del sistema, el rol del actor define la manera que este se relaciona o interactúa con el sistema. El nombre que se le dé a un actor deberá reflejar el papel que tendrá para el sistema.

Los Casos de Uso son una descripción de un conjunto de acciones que un sistema ejecuta y que produce un resultado de interés para un actor particular. Reflejan el uso que los actores le harán al sistema; con estos se muestran las funcionalidades que ofrecerá el sistema.

A continuación, se presenta los casos de uso del sistema en general:

Ilustración 3 Diagrama de Caso de Uso Ingreso al Sistema



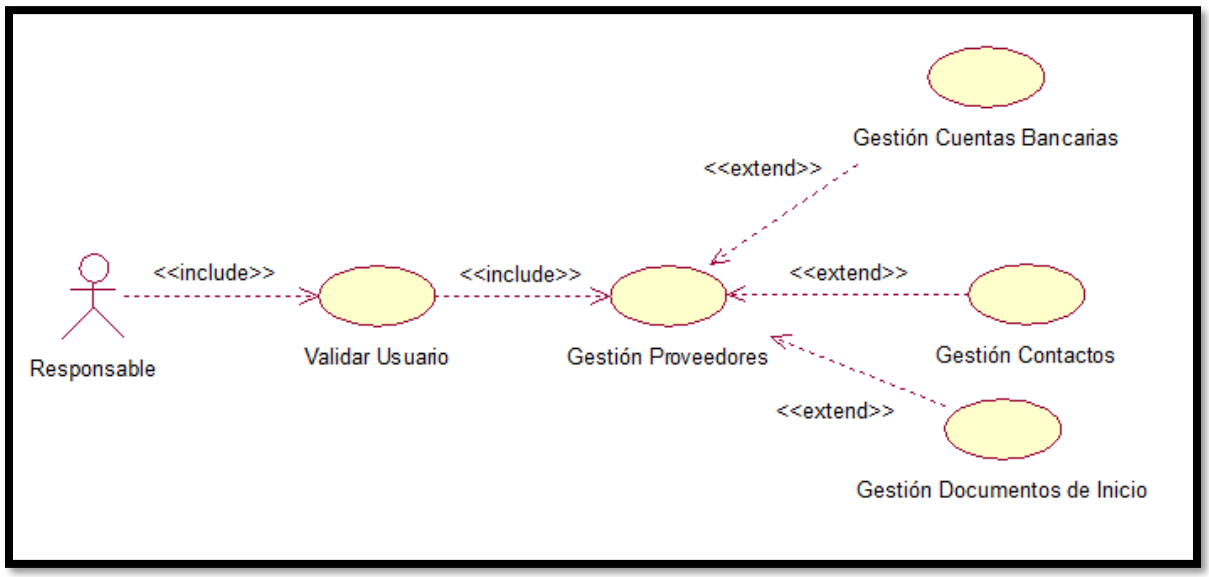
Representación del flujo para ingresar al sistema.

Tabla 3 Caso de Uso Ingreso al Sistema

Caso de Uso Ingreso al Sistema	
Caso de Uso	Ingreso al Sistema.
Actores	Responsable.
Propósito	Ingresar al Sistema.
Resumen	Es necesario que el usuario sea validado para poder ingresar al sistema. Valida al usuario mediante usuario (email) y contraseña.
Precondiciones	Tener un correo electrónico y contraseña para poder ingresar.
Flujo Principal	El usuario digita su email y contraseña. El sistema verifica si la información ingresada es correcta. El sistema permite el acceso siempre y cuando los datos sean correctos.

Tabla 3: Caso de Uso Ingreso al Sistema

Ilustración 4 Diagrama de Caso de Uso Acceso al Módulo de Proveedores



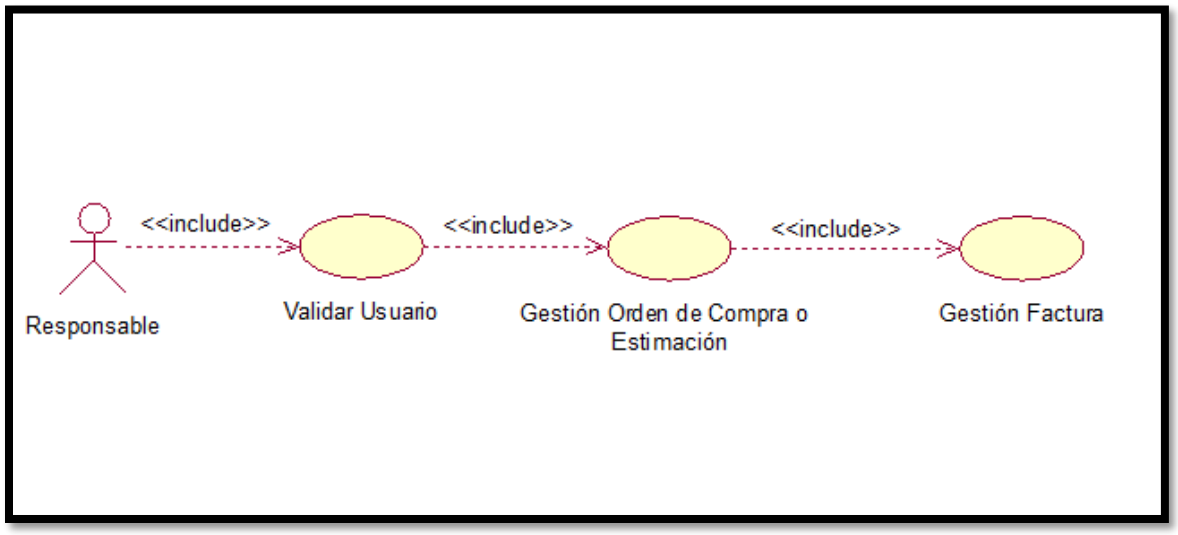
Representación del flujo para acceder al módulo de proveedores.

Tabla 4 Caso de Uso Acceso al Módulo de Proveedores

Caso de Uso Acceso al Módulo de Proveedores	
Caso de Uso	Acceso al Módulo de Proveedores.
Actores	Responsable.
Propósito	Permitir al responsable acceder al módulo de proveedores.
Resumen	El responsable puede gestionar toda la información dentro del módulo de proveedores.
Precondiciones	Tener el rol de responsable y tener acceso a la información del módulo de proveedores.
Flujo Principal	El usuario digita su email y contraseña. El sistema verifica si la información ingresada es correcta. El sistema permite el acceso siempre y cuando los datos sean correctos. El usuario puede gestionar toda la información del módulo de proveedores.

Tabla 4: Acceso al módulo de proveedores

Ilustración 5 Diagrama de Caso de Uso Acceso al Módulo de Orden de Compra o Estimación



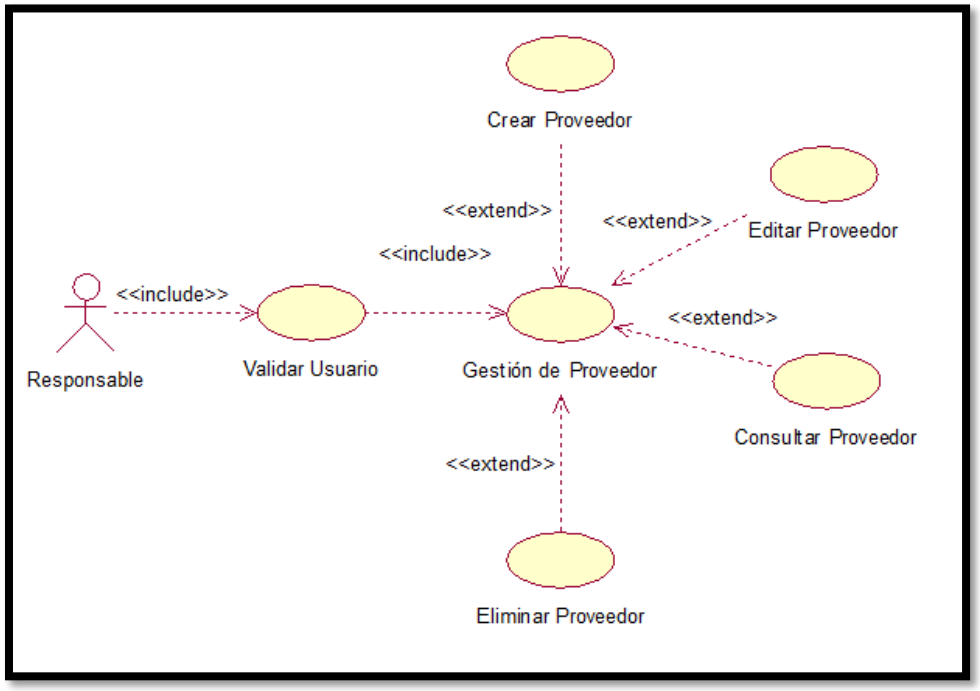
Representación del flujo para acceder al módulo orden de compra o estimación.

Tabla 5 Caso de Uso Acceso al Módulo de Orden de Compra o Estimación

Caso de Uso Acceso al Módulo de Orden de Compra o Estimación.	
Caso de Uso	Acceso al Módulo de Orden de Compra o Estimación.
Actores	Responsable.
Propósito	Permitir al responsable acceder al módulo de orden de compra o estimación.
Resumen	El responsable puede gestionar toda la información dentro del módulo de orden de compra o estimación.
Precondiciones	Tener el rol de responsable y tener acceso a la información del módulo de orden de compra o estimación.
Flujo Principal	El usuario digita su email y contraseña. El sistema verifica si la información ingresada es correcta. El sistema permite el acceso siempre y cuando los datos sean correctos. El usuario puede gestionar toda la información del módulo de orden de compra o estimación.

Tabla 5: Acceso al Módulo de Orden de Compra o Estimación.

Ilustración 6 Diagrama de Caso de Uso Gestión de Proveedor



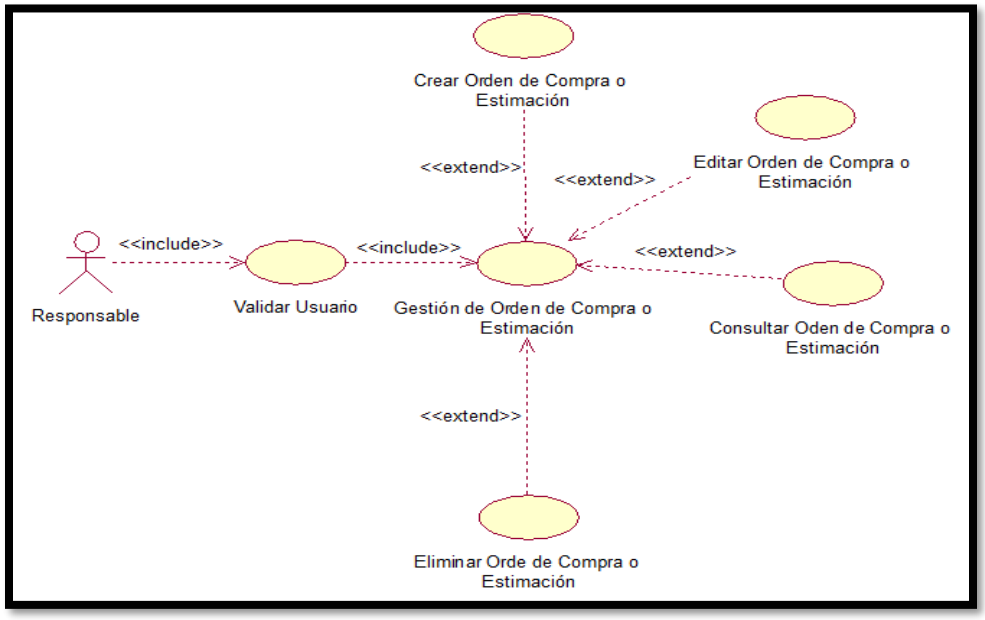
Representación del flujo para gestionar la información del proveedor.

Tabla 6 Caso de Uso Gestión de Proveedor

Caso de Uso Gestión de Proveedor	
Caso de Uso	Gestión de Proveedor.
Actores	Responsable.
Propósito	Permitir al responsable gestionar la información del proveedor.
Resumen	El responsable puede gestionar la información del proveedor que se da de alta.
Precondiciones	Tener el rol de responsable y tener acceso a la información del proveedor
Flujo Principal	El usuario digita su email y contraseña. El sistema verifica si la información ingresada es correcta. El sistema permite el acceso siempre y cuando los datos sean correctos. El usuario puede crear, editar, consultar y eliminar proveedores.

Tabla 6: Gestión de la información del proveedor.

Ilustración 7 Diagrama de Caso de Uso Gestión de Orden de Compra o Estimación



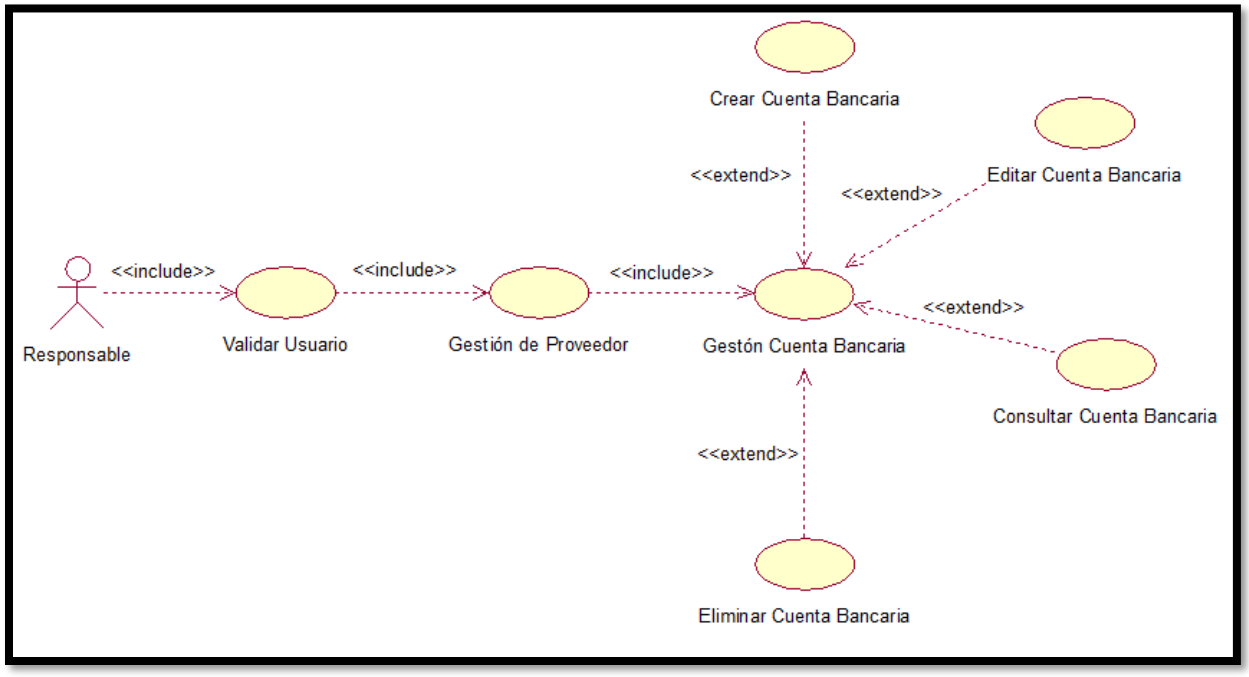
Representación del flujo para gestionar la información de orden de compra o estimación.

Tabla 7 Caso de Uso Gestión de Orden de Compra o Estimación

Caso de Uso Gestión de Orden de Compra o Estimación.	
Caso de Uso	Gestión de Orden de Compra o Estimación.
Actores	Responsable.
Propósito	Permitir al responsable gestionar la información de la orden de compra o estimación.
Resumen	El responsable puede gestionar la información de la orden de compra o estimación que se da de alta.
Precondiciones	Tener el rol de responsable y tener acceso a la información de la orden de compra o estimación.
Flujo Principal	El usuario digita su email y contraseña. El sistema verifica si la información ingresada es correcta. El sistema permite el acceso siempre y cuando los datos sean correctos. El usuario puede crear, editar, consultar y eliminar órdenes de compra o estimaciones.

Tabla 7: Gestión de la información de orden compra o estimación.

Ilustración 8 Diagrama de Caso de Uso Gestión de Cuenta Bancaria



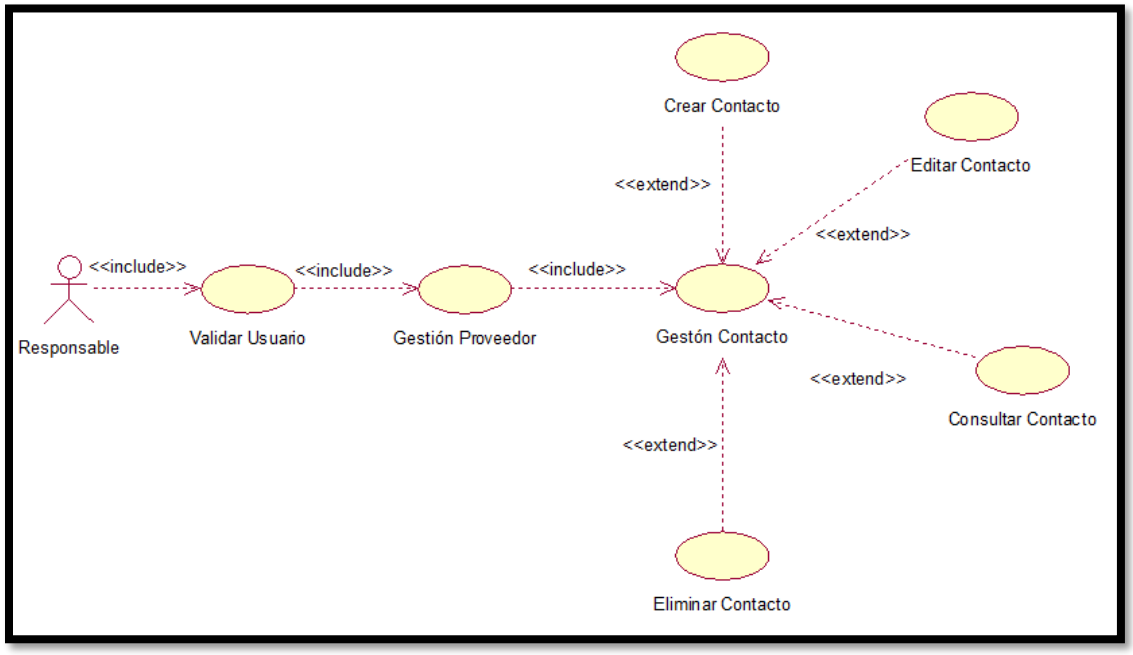
Representación del flujo para gestionar la información de cuenta bancaria.

Tabla 8 Caso de Uso Gestión de Cuenta Bancaria

Caso de Uso Gestión de Cuenta Bancaria	
Caso de Uso	Gestión de Cuenta Bancaria.
Actores	Responsable.
Propósito	Permitir al responsable gestionar la información de cuenta bancaria.
Resumen	El responsable puede gestionar la información de la cuenta bancaria que se da de alta.
Precondiciones	Tener el rol de responsable y tener acceso a la información de la cuenta bancaria.
Flujo Principal	El usuario digita su email y contraseña. El sistema verifica si la información ingresada es correcta. El sistema permite el acceso siempre y cuando los datos sean correctos. El usuario puede crear, editar, consultar y eliminar cuentas bancarias.

Tabla 8: Gestión de la información de cuenta bancaria.

Ilustración 9 Diagrama de Caso de Uso Gestión de Contacto



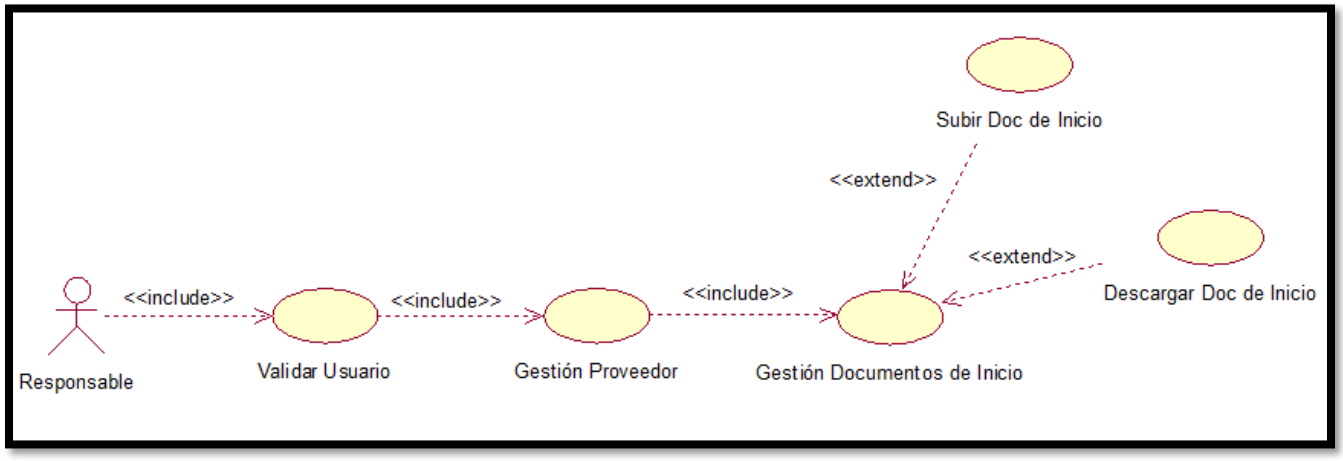
Representación del flujo para gestionar la información del contacto.

Tabla 9 Caso de Uso Gestión de Contacto

Caso de Uso Gestión de Contacto	
Caso de Uso	Gestión de Contacto.
Actores	Responsable.
Propósito	Permitir al responsable gestionar la información del contacto.
Resumen	El responsable puede gestionar la información del contacto que se da de alta.
Precondiciones	Tener el rol de responsable y tener acceso a la información del contacto
Flujo Principal	El usuario digita su email y contraseña. El sistema verifica si la información ingresada es correcta. El sistema permite el acceso siempre y cuando los datos sean correctos. El usuario puede crear, editar, consultar y eliminar contactos.

Tabla 9: Gestión de la información del contacto.

Ilustración 10 Diagrama de Caso de Uso Gestión de Documentos de Inicio



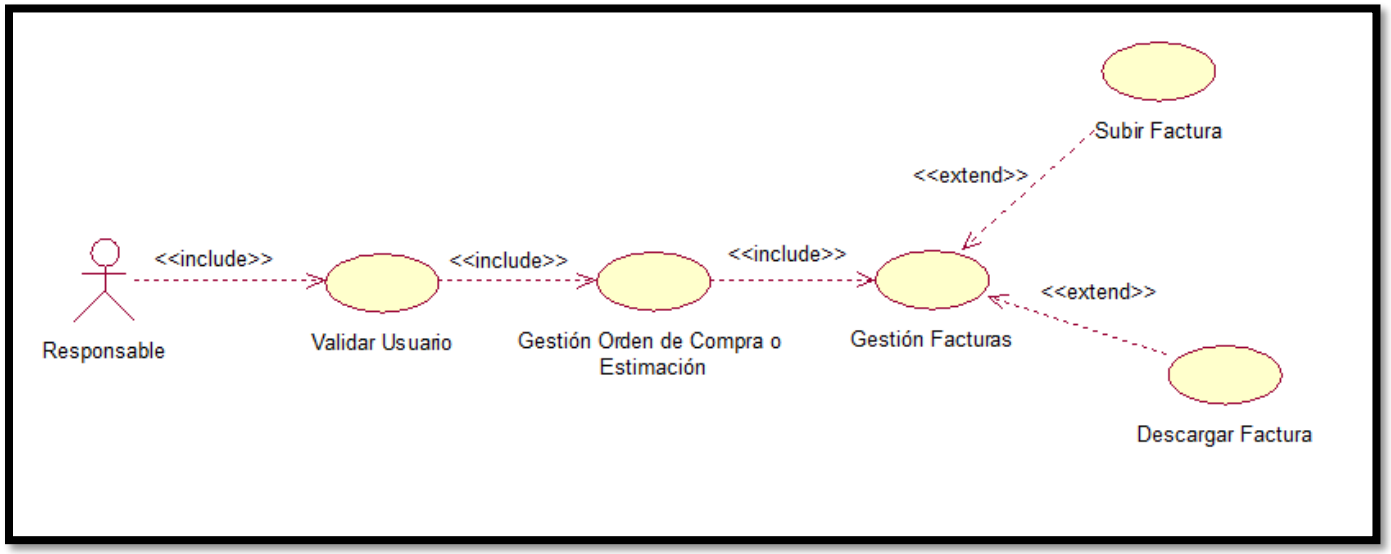
Representación del flujo para gestionar los documentos de inicio.

Tabla 10 Caso de Uso Gestión de Documentos de Inicio

Caso de Uso Gestión de Documentos de Inicio	
Caso de Uso	Gestión de Documentos de Inicio.
Actores	Responsable.
Propósito	Permitir al responsable gestionar documentos de inicio.
Resumen	El responsable puede subir o descargar documentos de inicio.
Precondiciones	Tener el rol de responsable y tener acceso al apartado documentos de inicio.
Flujo Principal	El usuario digita su email y contraseña. El sistema verifica si la información ingresada es correcta. El sistema permite el acceso siempre y cuando los datos sean correctos. El usuario puede subir y descargar documentos de inicio.

Tabla 10: Gestión de los documentos de inicio.

Ilustración 11 Diagrama de Caso de Uso Gestión de Facturas



Representación del flujo para gestionar las facturas.

Tabla 11 Caso de Uso Gestión de Facturas

Caso de Uso Gestión de Facturas	
Caso de Uso	Gestión de Facturas.
Actores	Responsable.
Propósito	Permitir al responsable gestionar facturas.
Resumen	El responsable puede subir o descargar facturas.
Precondiciones	Tener el rol de responsable y tener acceso al apartado de facturas.
Flujo Principal	El usuario digita su email y contraseña. El sistema verifica si la información ingresada es correcta. El sistema permite el acceso siempre y cuando los datos sean correctos. El usuario puede subir y descargar facturas.

Tabla 11: Gestión de las facturas.

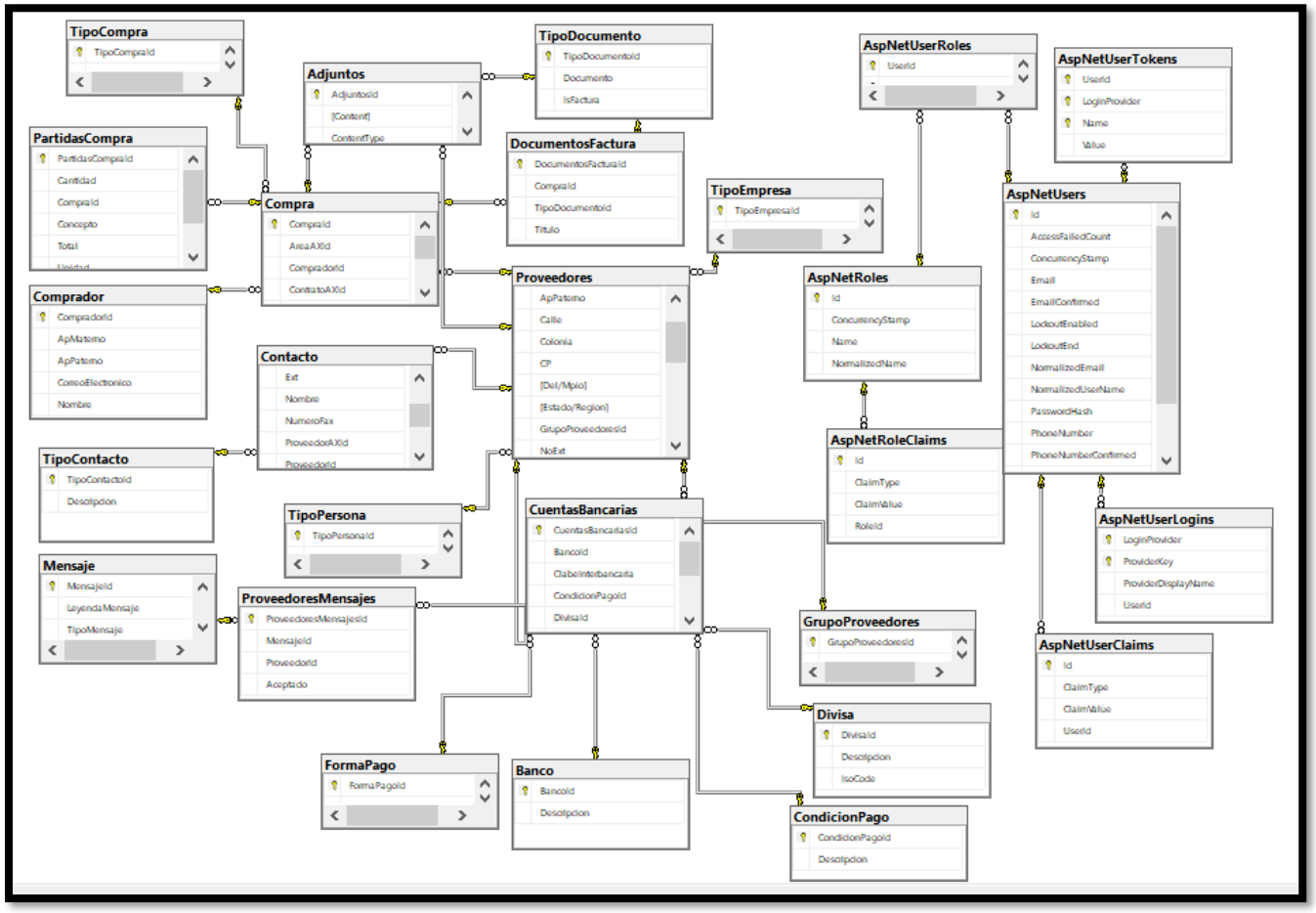
5.2 Fase de análisis y diseño

Es la creación apropiada de módulos de programas y de interfaces, para implementar la especificación creada en la fase anterior. Es decir, esta fase muestra la relación entre el diseño de las bases de datos y el diseño de módulos funcionales (pantallas o vistas).

5.2.1 Creación de modelo Entidad Relación

En la siguiente ilustración se presenta el modelo entidad relación del sistema, en él se reflejan las principales entidades y las relaciones que existen entre ellas.

Ilustración 12 Modelo Entidad Relación

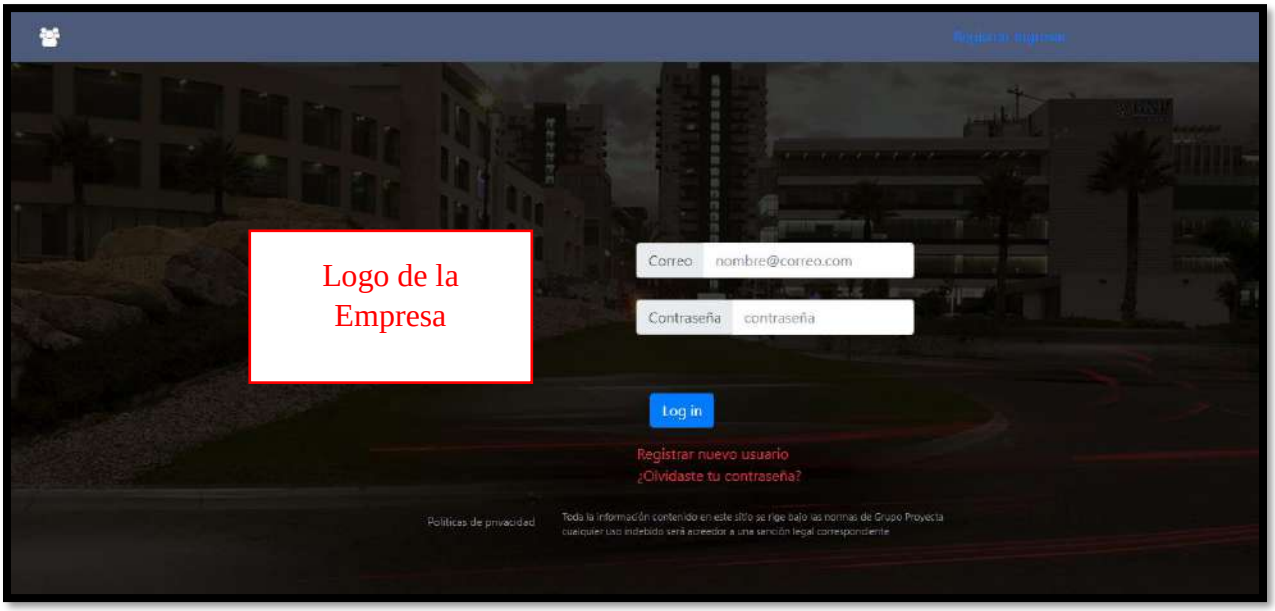


Representación de las entidades y relaciones de la Base de Datos.

5.2.3 Creación de las vistas de usuarios

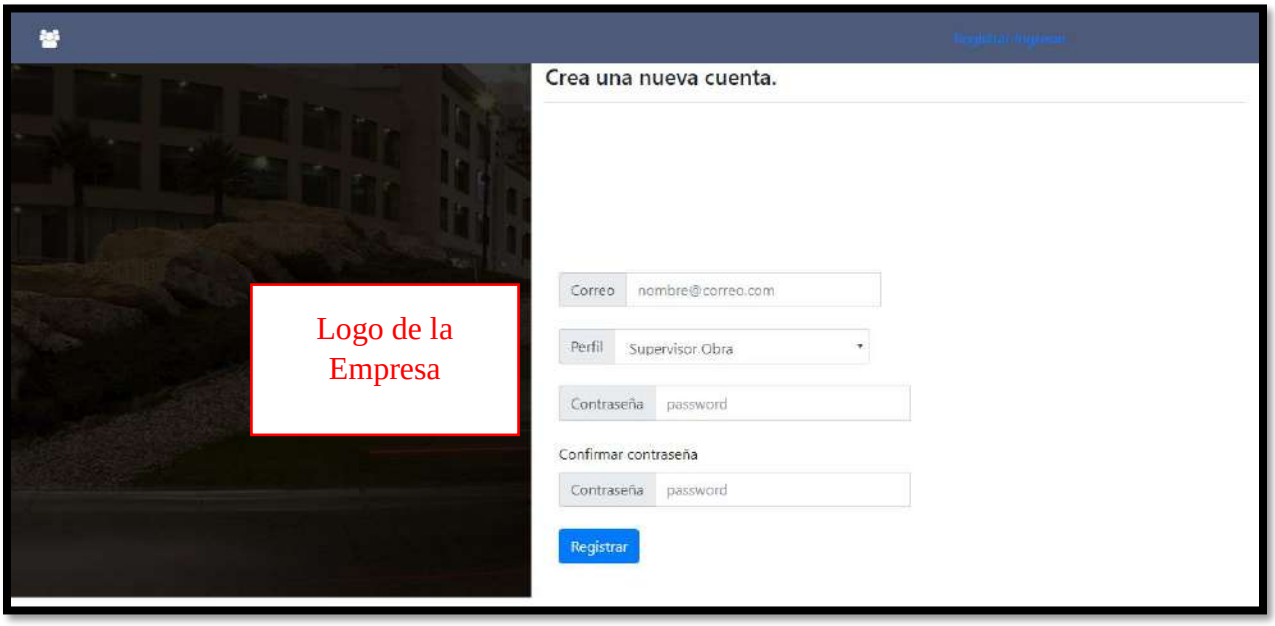
En este apartado se detalla el diseño preliminar del Sistema, dentro de esta etapa se diseñan las pantallas de usuarios que se usarán en la interacción del usuario final con la aplicación. Esta fase debe realizarse en conjunto con el usuario, para identificar qué es lo que más se acomoda a su gusto y prioridad. A continuación, se presentan algunas de las vistas del Sistema.

Ilustración 13 Vista Login



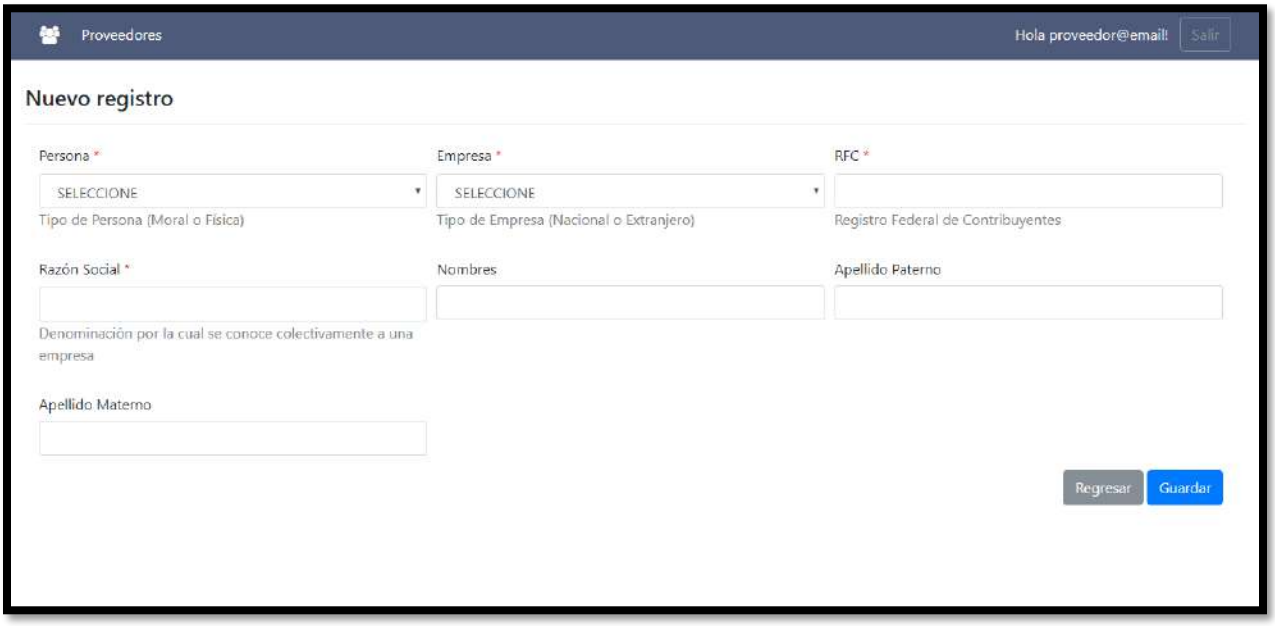
Representación de la vista login.

Ilustración 14 Vista Crear Cuenta de Acceso



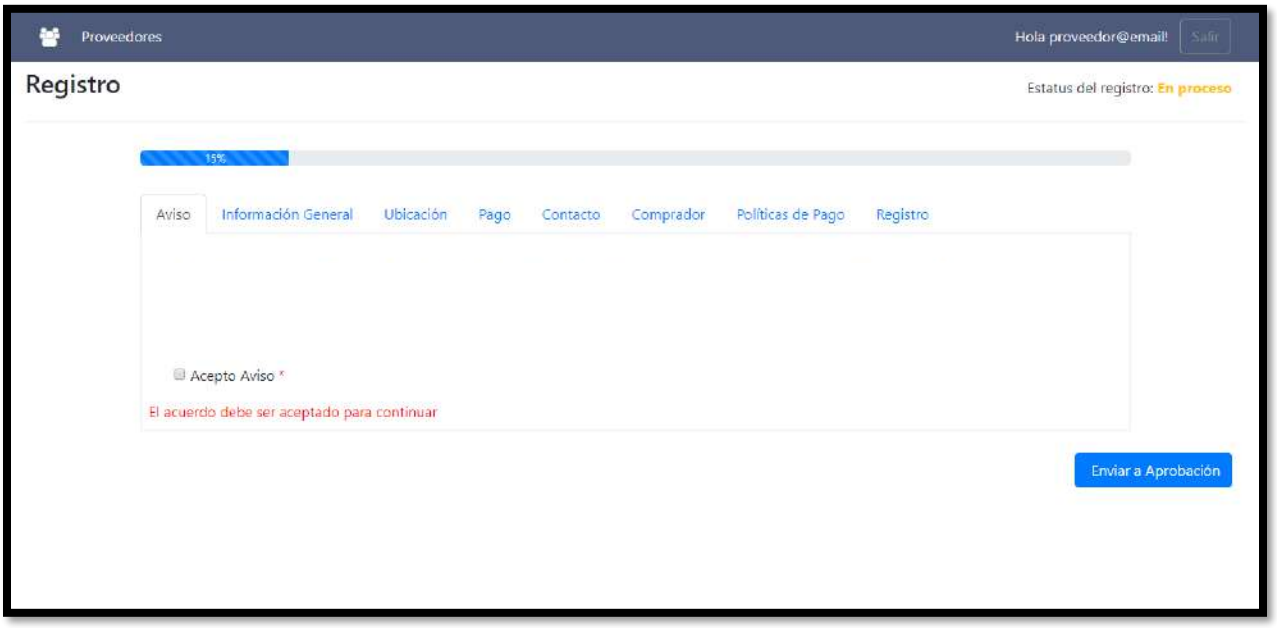
Representación de la vista crear cuenta de acceso.

Ilustración 15 Vista Alta Proveedor



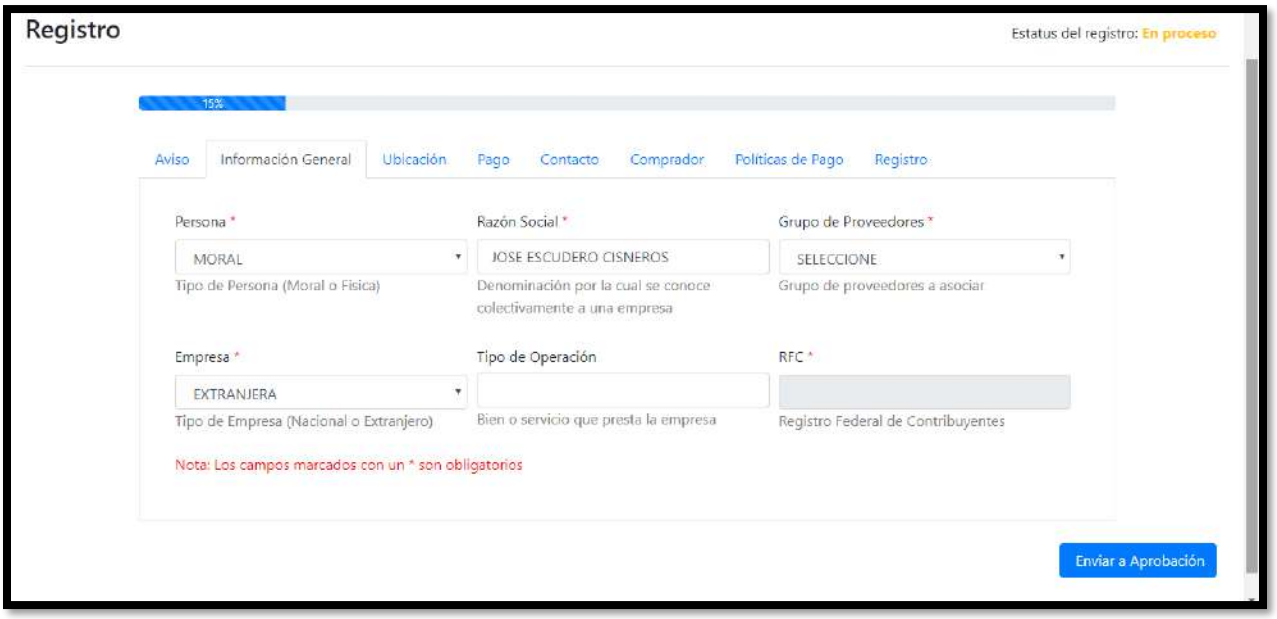
Representación de la vista dar alta nuevo proveedor

Ilustración 16 Vista Aceptar Aviso



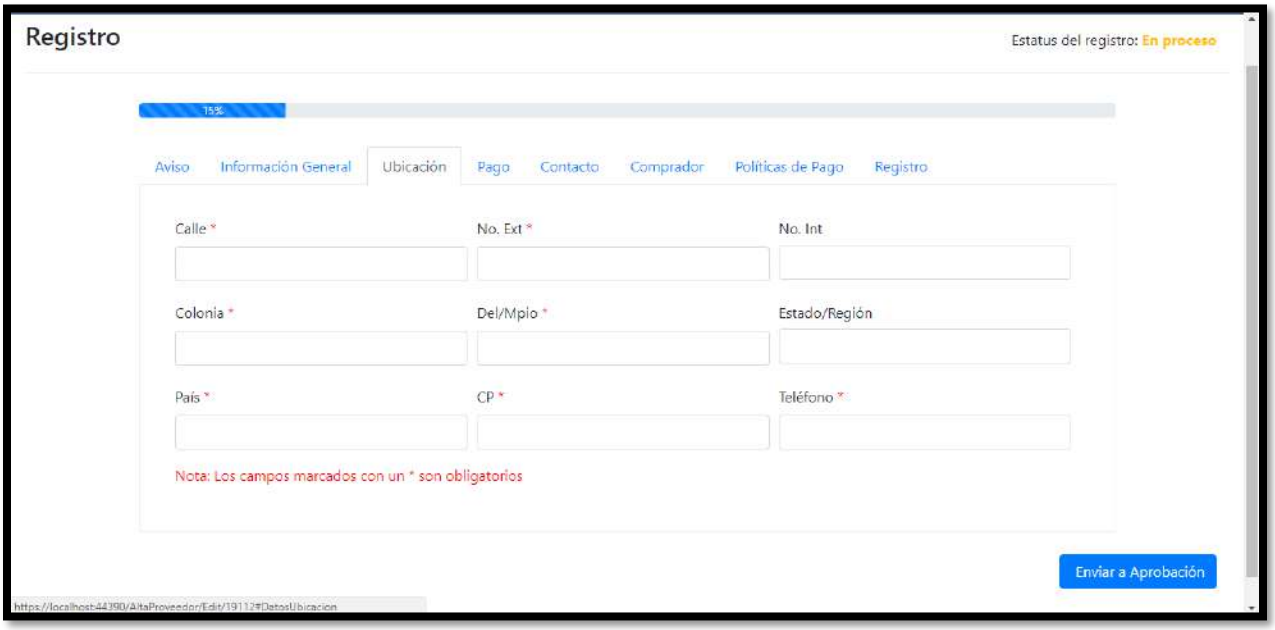
Representación de la vista aceptar aviso de privacidad.

Ilustración 17 Vista Datos Generales



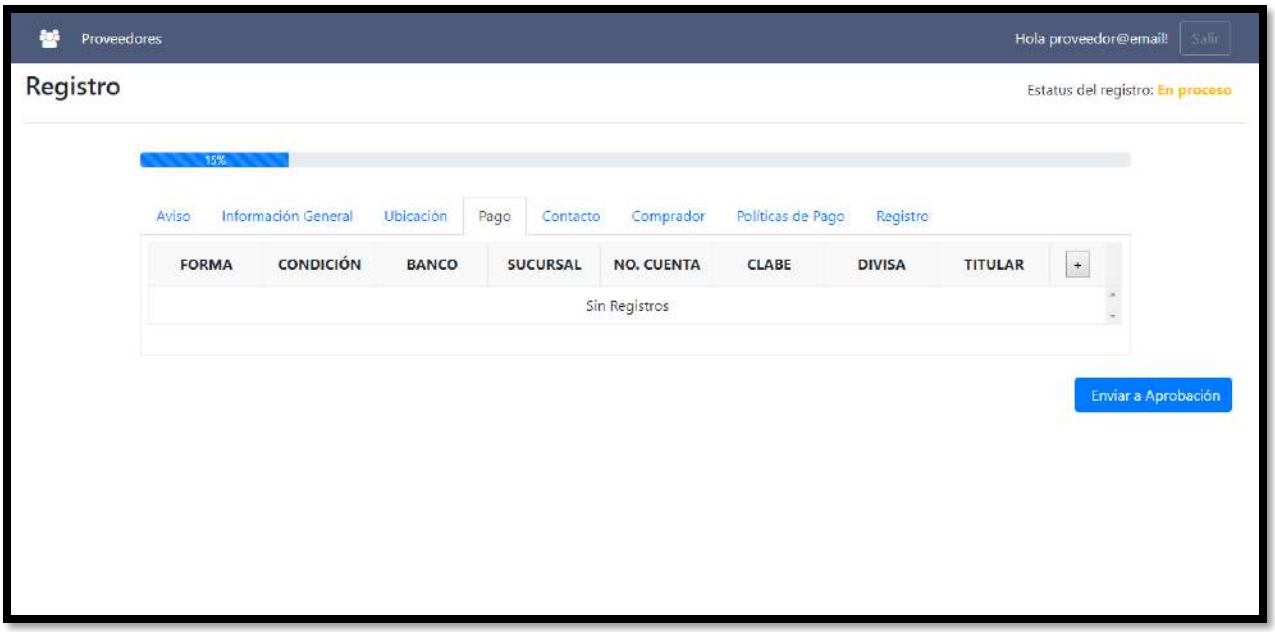
Representación de la vista datos generales del proveedor.

Ilustración 18 Vista Datos de Ubicación



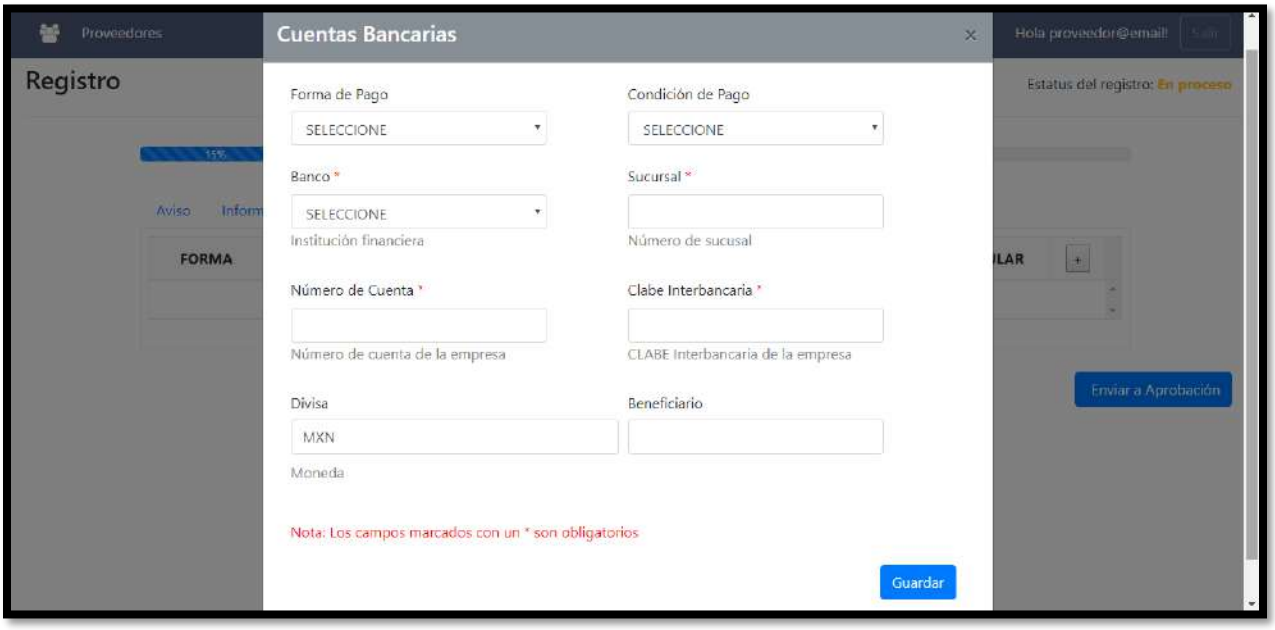
Representación de la vista datos de ubicación del proveedor.

Ilustración 19 Vista Grid Datos de Pago



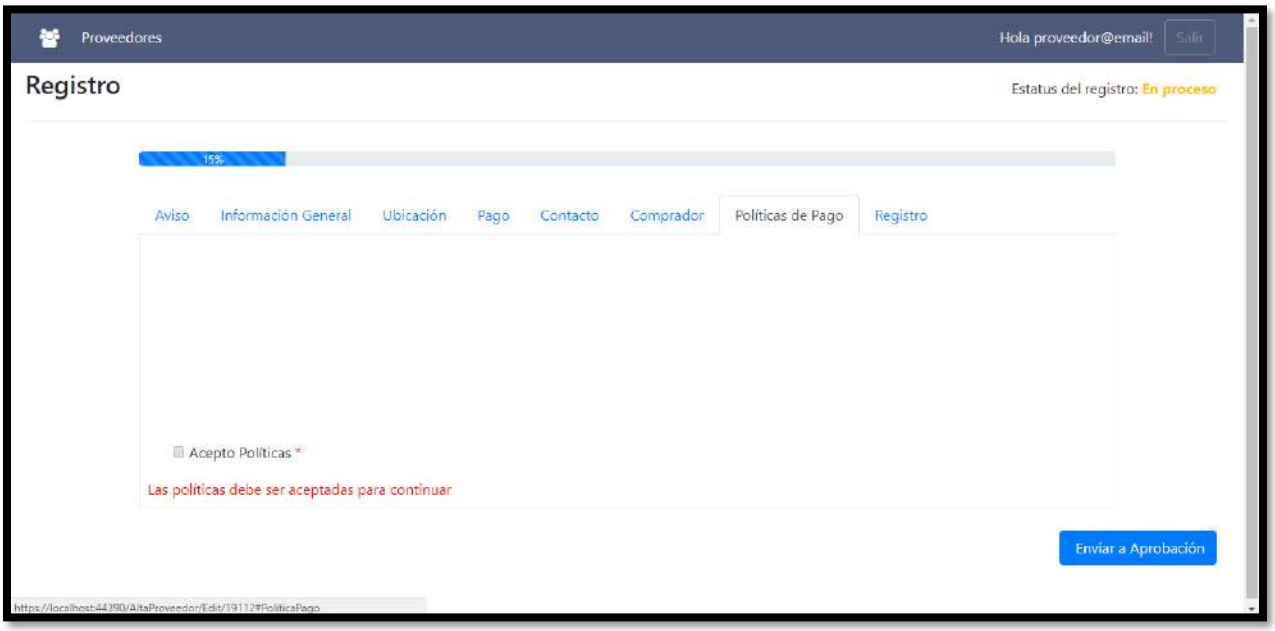
Representación de la vista en donde se muestra el grid de los datos de pago del proveedor.

Ilustración 20 Vista Alta Cuenta Bancaria



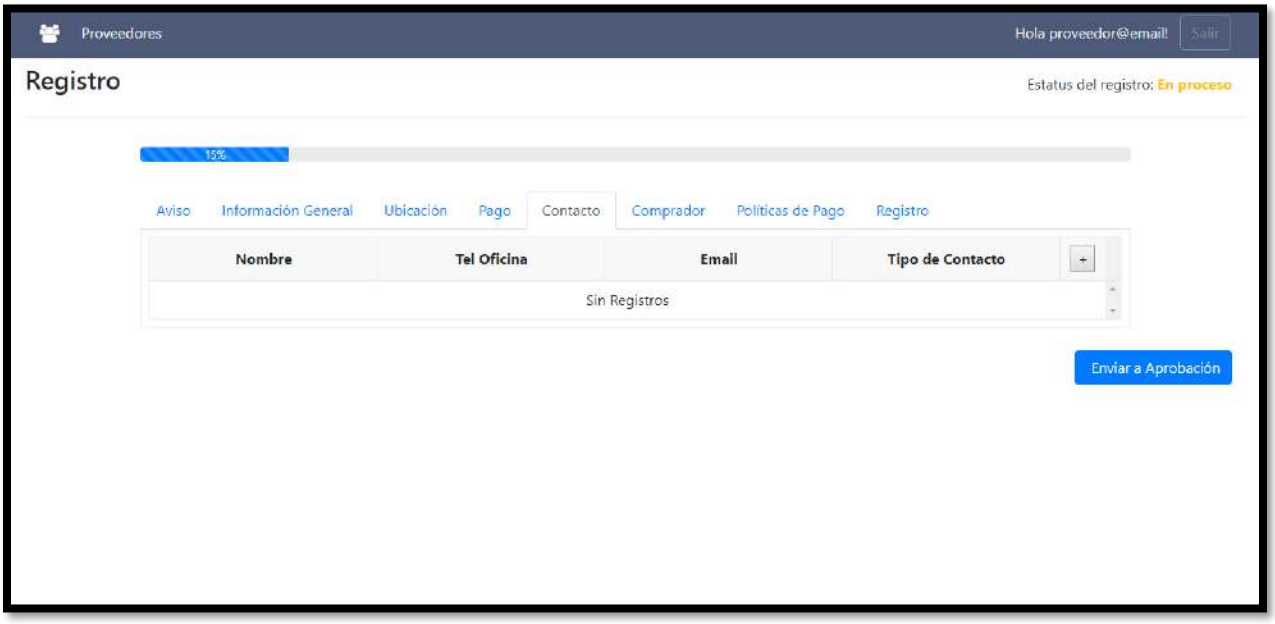
Representación de la vista dar de alta cuenta bancaria.

Ilustración 21 Vista Aceptar Políticas



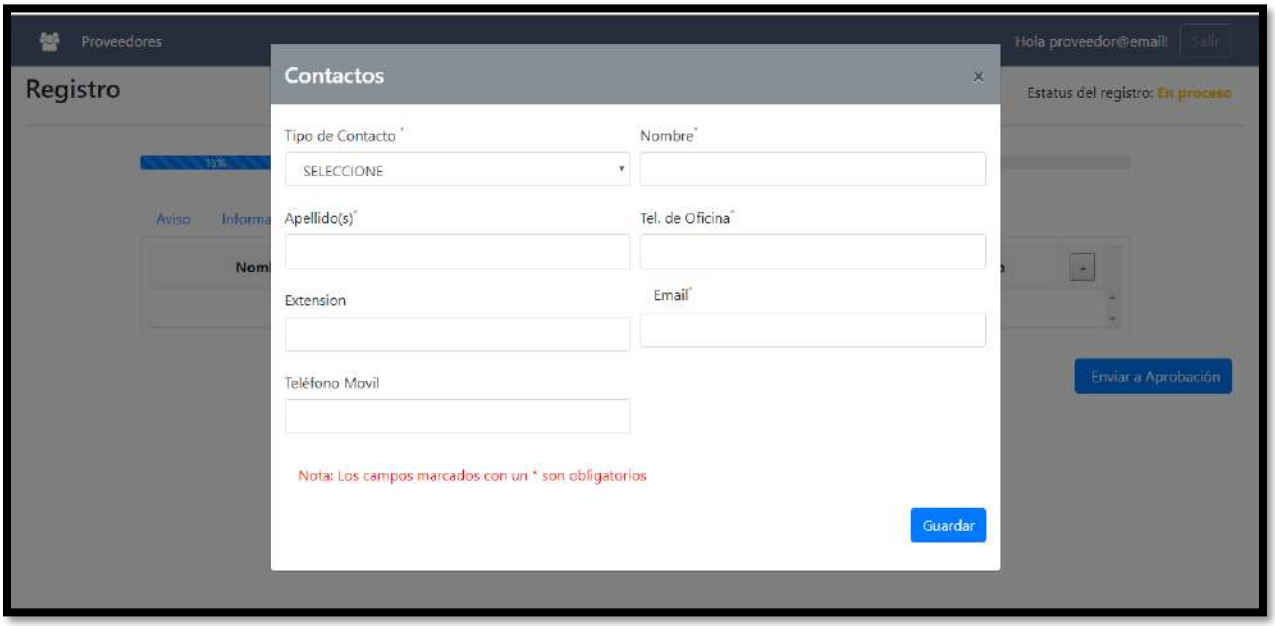
Representación de la vista aceptar políticas.

Ilustración 22 Vista Grid Contactos



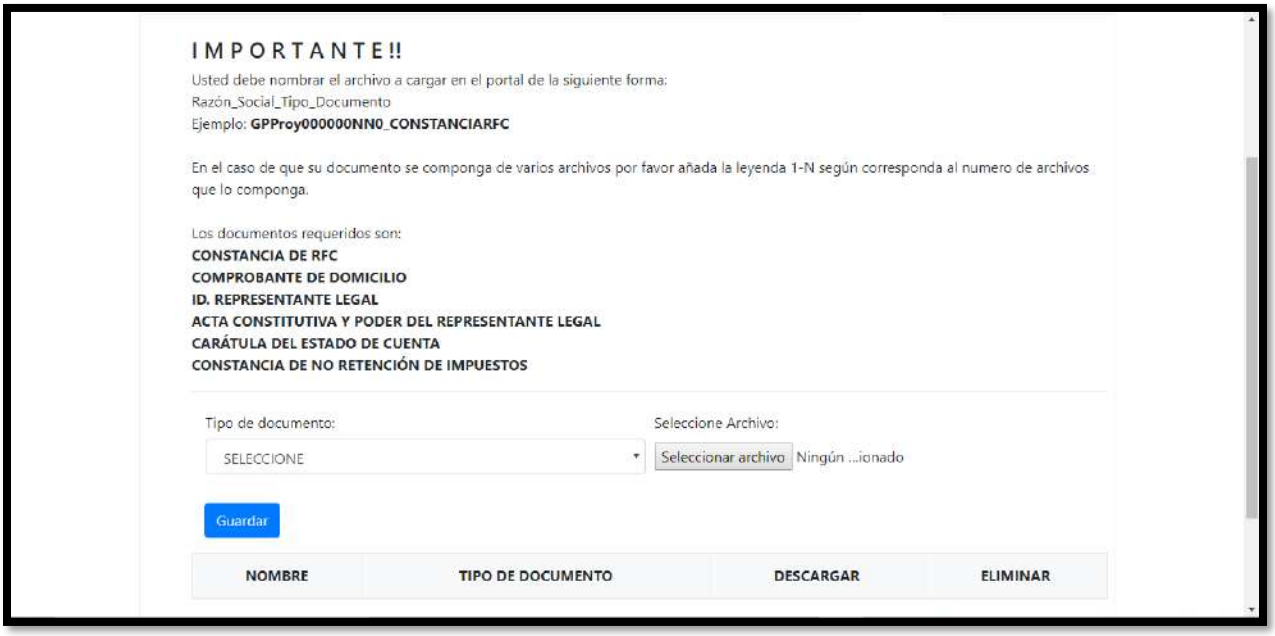
Representación de la vista en donde se muestra el grid de contactos del proveedor.

Ilustración 23 Vista Alta Contacto



Representación de la vista para dar de alta contactos del proveedor.

Ilustración 24 Vista Documentos de Inicio



Representación de la vista para dar de alta los documentos de registro del proveedor.

5.3 Etapa de codificación y construcción

5.3.1 Marco de trabajo

Este apartado se muestra el marco de creación del sistema para una empresa inmobiliaria. Como se mencionó anteriormente, la idea es desarrollar una aplicación Web bajo la plataforma de trabajo ASP.NET, por lo cual, se ha optado por una versión de ASP.NET basada en MVC.

Una vez elegida la plataforma se definen a continuación los elementos necesarios para crear una aplicación en ASP.NET MVC3.

5.3.2 Creación de un proyecto básico en asp.net mvc3

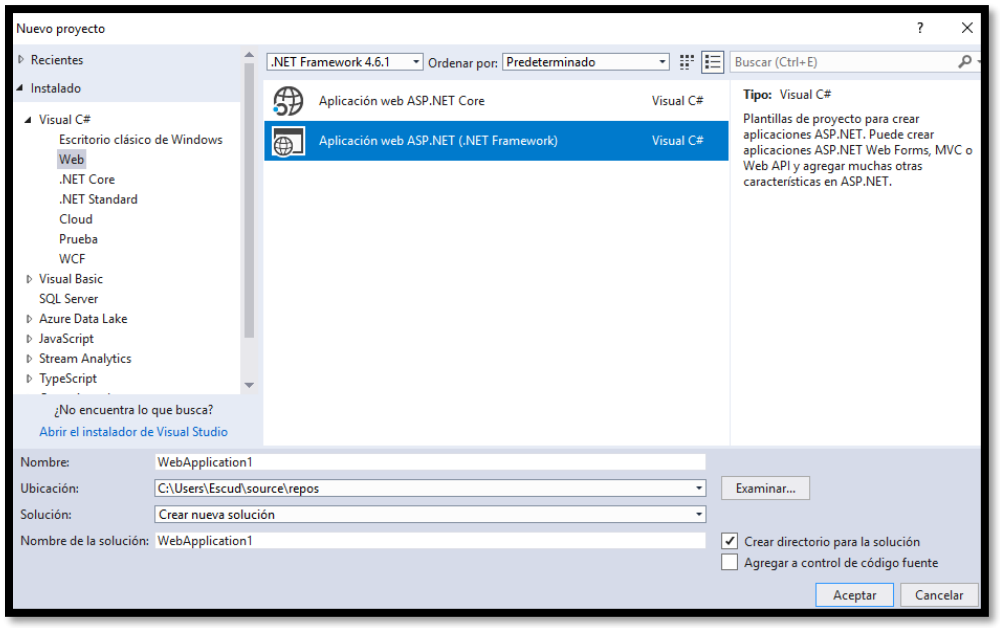
Para desarrollar e implementar una aplicación en ASP.NET MVC3 es necesario tener instalado la herramienta de trabajo Visual Studio 2017. A continuación, se muestra cómo crear

una aplicación MVC3 y todos los elementos que lo componen, pero antes de comenzar se necesita descargar la plataforma ASP.NET MVC3.

5.3.2.1 Proyecto mvc3

Se procede a abrir un nuevo proyecto para esto se elige el menú de Archivo, después Nuevo y se elige Proyecto; se selecciona la opción de Web, posteriormente se elige Aplicación web de ASP.NET (.NET Framework). En la parte de abajo se indica el nombre del proyecto y dónde se quiere almacenar. Por último, se pulsa Aceptar.

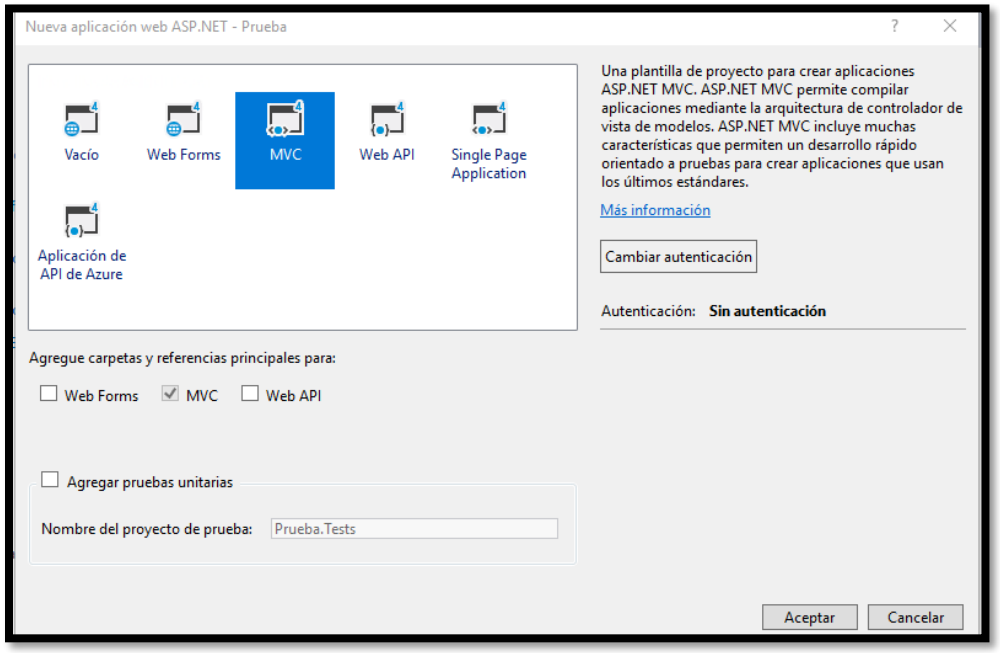
Ilustración 25 Pantalla elección de la aplicación



Representación de pantalla para elección de tipo de aplicación.

Posteriormente se selecciona la opción MVC, dejando el resto de los valores por defecto. Se pulsa nuevamente Aceptar.

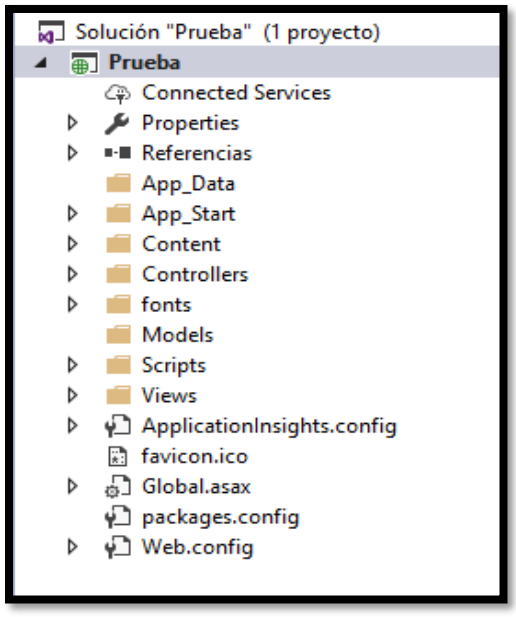
Ilustración 26 Pantalla elección de Plantilla



Representacion de pantalla para elección de plantilla.

Seguido de esto se verá un proyecto con la siguiente estructura.

Ilustración 27 Pantalla representación de la Solución

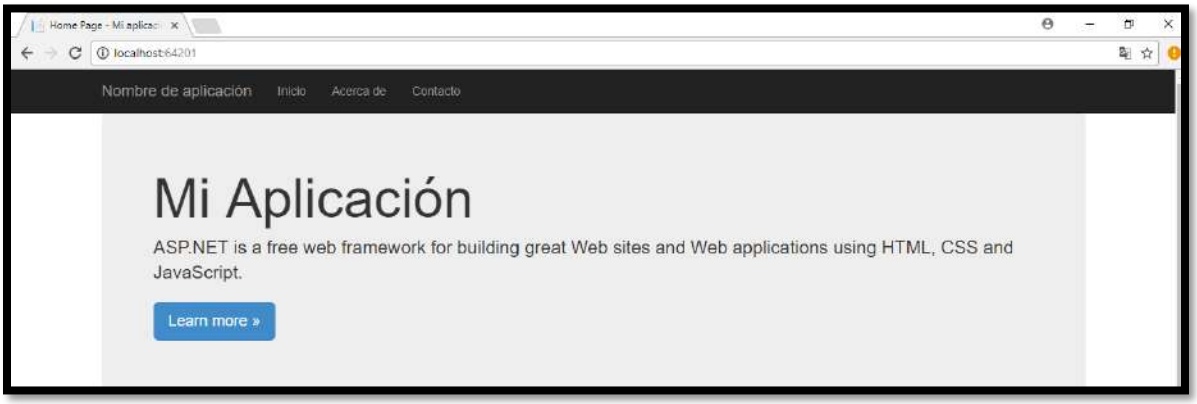


Representación del diagrama de la solución.

- Content: Carpeta de almacenamiento del contenido estático de la aplicación. Generalmente se alojan las hojas de estilo (CSS), las imágenes, etc.
- Controllers: Carpeta de almacenamiento de los controladores de la aplicación. El marco MVC requiere que los nombres de todos los controladores terminen con "Controller", ejemplos: HomeController, UsuarioController, BancoController, entre otros.
- Models: Carpeta de almacenamiento de las clases que representan los modelos que se usarán en la aplicación.
- Scripts: Carpeta de almacenamiento dónde se alojan todos los ficheros JavaScript que se necesitan en la aplicación. Por defecto ya encuentra un conjunto de ficheros que se usarán para las llamadas AJAX y la biblioteca de JQuery (que es la que usa por defecto Visual Studio).
- Views: Carpeta de almacenamiento en donde se ubican todas las vistas del sistema. La carpeta View contiene muchas carpetas, esto se produce al generar una vista, esta crea una carpeta con el nombre del controlador (quitando la palabra "Controller"), y a su vez dentro de ésta, se crearán tantos ficheros como acciones disponga el controlador. Dentro de la carpeta Views se encuentra una carpeta denominada Shared que se usa para alojar todo aquello que sea común para los controladores.

Para probar si la aplicación que se creó funciona correctamente, se pulsa la tecla F5 o la imagen de reproducción y se abrirá el navegador por defecto, con una página similar a la siguiente.

Ilustración 28 Pantalla representación de la ejecución de la aplicación

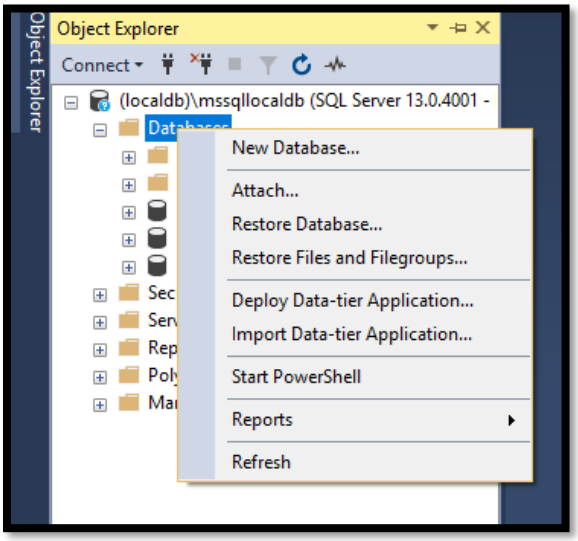


Representación de la ejecución de la aplicación.

5.3.2.1.1 Creación de la base de datos

En este apartado se muestra cómo crear la base de datos que se usará en el sistema, para ello se necesita tener instalado SQL Server 2017. Primeramente, se inicia el Manejador de base de datos, se autentica y posteriormente se dirige al explorador de objetos. Se da clic derecho en la carpeta Base de Datos y se elige Nueva Base de Datos.

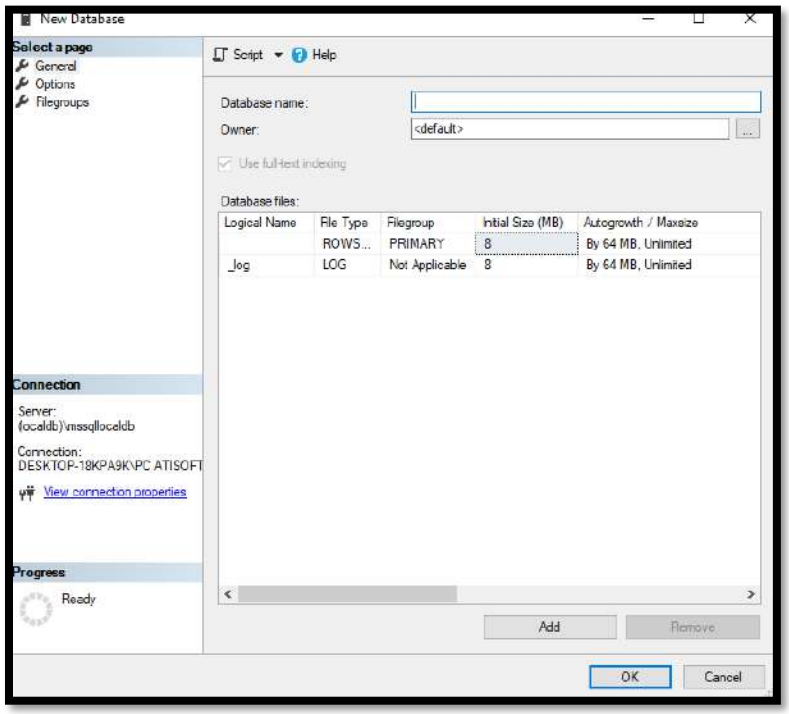
Ilustración 29 Pantalla Creación de la Base de Datos



Representación de la creación de un BD.

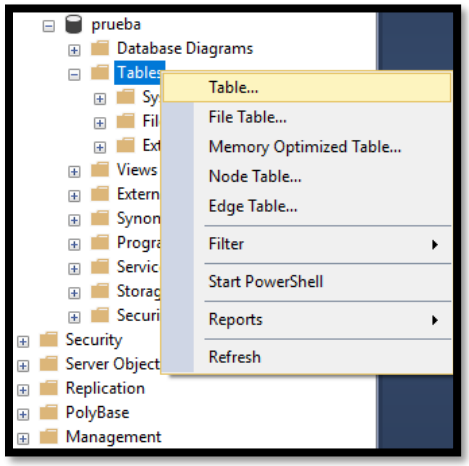
Aparecerá una ventana como la que se muestra a continuación en donde se tiene que colocar un nombre para la base de datos, después de asignar el nombre se da clic en OK.

Ilustración 30 Pantalla asignar nombre a la Base de Datos



Representación de pantalla asignación de nombre a la base de datos.

Ilustración 31 Pantalla creación de una Tabla



Representación de la creación de una Tabla.

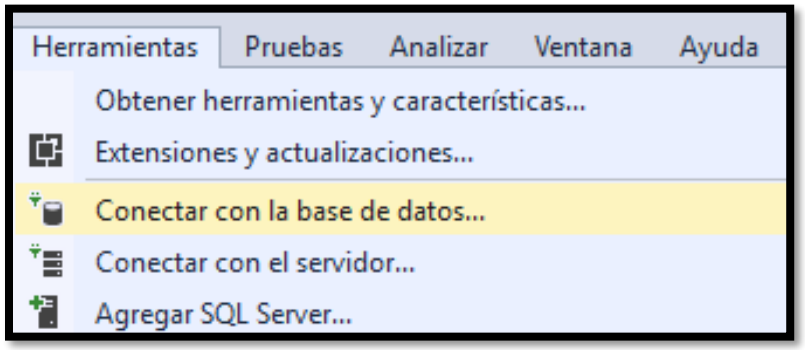
Una vez creada la Base de datos se procede a crear cada una de las tablas que el sistema necesita, primeramente, en el explorador de objetos se expande la carpeta de nombre Base de Datos, después se busca la base de datos que se creó y se da clic derecho a la carpeta Tablas. Aparecerá un apartado en donde se debe de especificar los campos o atributos que tendrá la tabla, así como también el tipo de datos y sus restricciones.

5.3.2.1.2 Conexión a la base de datos desde Visual Studio

A continuación, se darán a conocer los pasos necesarios para crear los modelos, en el proyecto (solución) de Visual Studio a partir de la base de datos que se creó en SQL Server. Con esta conexión los datos que se capturen en la aplicación se almacenarán en la base de datos de SQL Server.

Lo primero que se necesita hacer es iniciar Visual Studio, ir al menú Herramientas y elegir la opción Conectar con la base de datos.

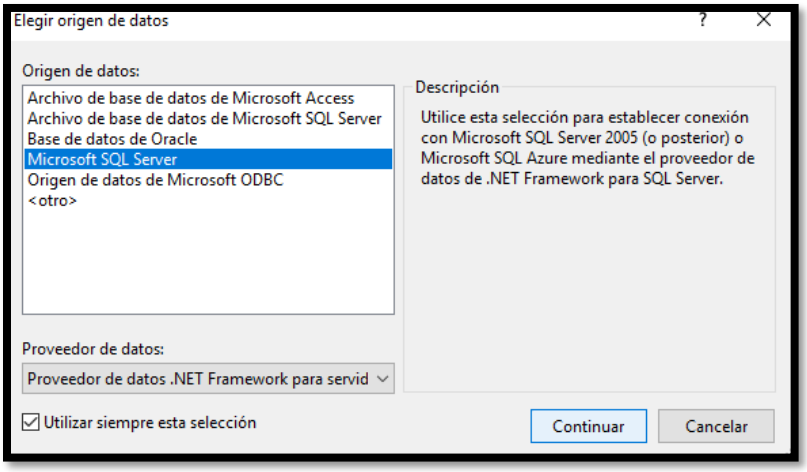
Ilustración 32 Pantalla Conectar con la Base de Datos



Representación de elegir opción Conectar con la base de datos.

Aparecerá otra pantalla en donde se tiene que indicar la procedencia de los datos, en este caso se elige Microsoft SQL Server.

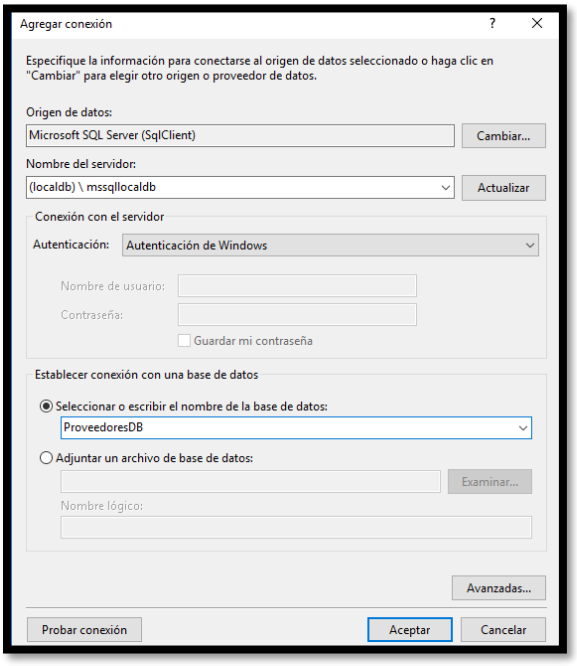
Ilustración 33 Pantalla elección de la procedencia de datos



Representación de elegir opción origen de datos.

Siguiente paso es colocar el nombre del servidor al que se desea conectar y el nombre de la base de datos a la que se hará referencia.

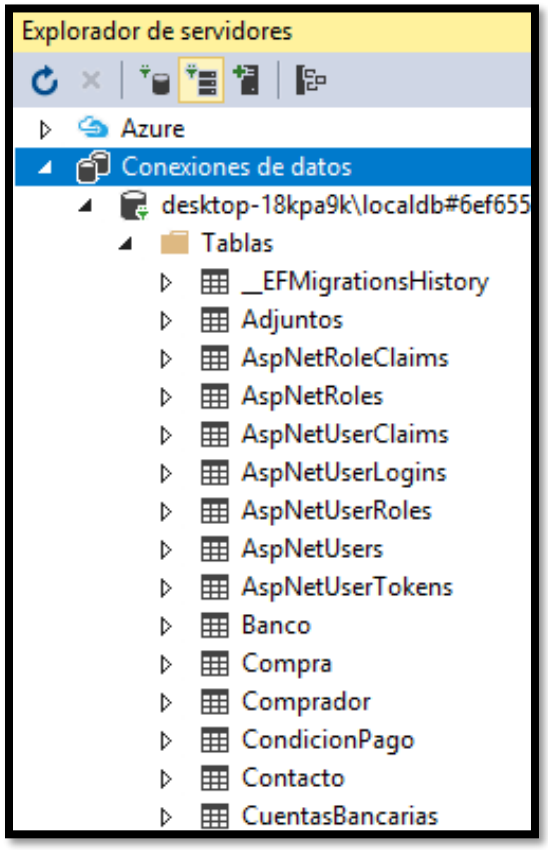
Ilustración 34 Pantalla indicar nombre del Servidor y nombre de la Base de Datos



Representación de elegir nombre del servidor y nombre de la BD.

Ahora la base de datos ya está instanciada en el proyecto, para comprobarlo basta con dirigirse al explorador de servidores, elegir conexión de datos y se mostrarán todas las tablas pertenecientes a la base de datos.

Ilustración 35 Pantalla comprobación de la conexión con la Base de Datos



Representación de la conexión exitosa con la BD.

El siguiente paso es instalar Entity Framework Core, para instalarlo primero es dirigirse al menú Herramientas, después elegir NuGet Package Manager y por último elegir Consola del gestor de paquetes. En la consola colocar la siguiente instrucción y dar enter.

```
Install-Package Microsoft.EntityFrameworkCore.SqlServer
```

También se usarán herramientas de Entity Framework para crear un modelo de la base de datos. Así que se instalará el siguiente paquete de herramientas:

```
Install-Package Microsoft.EntityFrameworkCore.Tools
```

Se usarán algunas herramientas de ASP.NET Core para crear controladores y vistas. Así que se instalará el siguiente paquete de diseño:

```
Install-Package Microsoft.VisualStudio.Web.CodeGeneration.Design
```

A continuación, se creará el modelo Entity Framework con la base de datos existente. Para esto es necesario ejecutar el siguiente comando:

```
Scaffold-DbContext
"Server=(localdb)\mssqllocaldb;Database=ProveedoresDB;Trusted_Connection=True;"
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models
```

Este procedimiento creó un contexto basado en el esquema de la base de datos, el contexto representa una sesión con la base de datos y le permite consultar y guardar instancias de las clases de entidad.

```
public partial class DbPortalProveedoresContext :
IdentityDbContext<ApplicationUser>
    {
        public virtual DbSet<Adjuntos> Adjuntos { get; set; }
        public virtual DbSet<DocFactura> DocFactura { get; set; }
        public virtual DbSet<DocProveedor> DocProveedor { get; set; }

        public virtual DbSet<Banco> Banco { get; set; }
        public virtual DbSet<Compra> Compra { get; set; }
        public virtual DbSet<Comprador> Comprador { get; set; }
        public virtual DbSet<CondicionPago> CondicionPago { get; set; }
        public virtual DbSet<Contacto> Contacto { get; set; }
        public virtual DbSet<CuentasBancarias> CuentasBancarias { get; set; }
        public virtual DbSet<Divisa> Divisa { get; set; }
        public virtual DbSet<DocumentosFactura> DocumentosFactura { get; set; }
        public virtual DbSet<EmpresaProyecta> EmpresaProyecta { get; set; }
        public virtual DbSet<FormaPago> FormaPago { get; set; }
        public virtual DbSet<GrupoProveedores> GrupoProveedores { get; set; }
        public virtual DbSet<PartidasCompra> PartidasCompra { get; set; }
        public virtual DbSet<Proveedores> Proveedores { get; set; }
        public virtual DbSet<TipoCompra> TipoCompra { get; set; }
        public virtual DbSet<TipoContacto> TipoContacto { get; set; }
        public virtual DbSet<TipoDocumento> TipoDocumento { get; set; }
        public virtual DbSet<TipoEmpresa> TipoEmpresa { get; set; }
        public virtual DbSet<TipoPersona> TipoPersona { get; set; }
        public virtual DbSet<Mensaje> Mensaje { get; set; }
        public virtual DbSet<ProveedoresMensajes> ProveedoresMensajes { get; set;
}
```

El concepto de inyección de dependencia es central para ASP.NET Core. Los servicios se registran con una inyección de dependencia durante el inicio de la aplicación. Los componentes que requieren estos servicios (como los controladores MVC) proporcionan estos servicios a través de parámetros de constructor o propiedades. El siguiente paso es eliminar la configuración de contexto en línea, en ASP.NET Core, la configuración generalmente se realiza en Startup.cs.

Es necesario eliminar el siguiente método.

```
protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
{
optionsBuilder.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=ProveedoresDB;Trusted_Connection=True;");
}
```

Y agregar el siguiente constructor, el cual permitirá pasar el contexto a la inyección de dependencia.

```
  public DbPortalProveedoresContext(DbContextOptions<DbPortalProveedoresContext> options)
          : base(options)
    { }
```

Ahora se registrará y configurará el contexto en Startup.cs como un servicio, para esto es necesaria agregar las siguientes líneas al principio del archivo.

```
using PortalProveedoresGP.Models;
using Microsoft.EntityFrameworkCore;
```

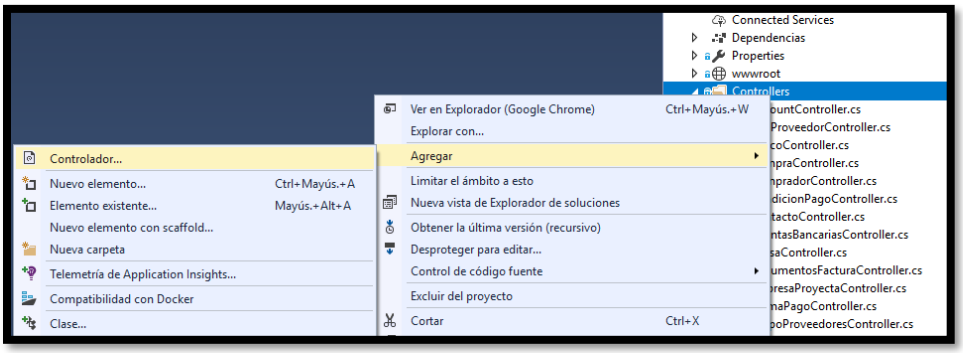
Por último, se busca el método ConfigureServices y se agregan las siguientes líneas de código para registrar el contexto como un servicio.

```
 public void ConfigureServices(IServiceCollection services)
    {
       services.AddDbContext<DbPortalProveedoresContext>(options =>
          options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

       services.AddIdentity<ApplicationUser, IdentityRole>()
         .AddEntityFrameworkStores<DbPortalProveedoresContext>()
         .AddDefaultTokenProviders();
       // Add application services.
       services.AddTransient<IEmailSender, EmailSender>();
       services.AddMvc();
    }
```

5.3.2.1.3 Generación de un Controlador (Controller)

El controlador es un elemento muy importante para el desarrollo de los proyectos, por medio de él se realiza la comunicación entre la base de datos y las vistas del usuario. Para crear un controlador lo primero que se tiene que hacer es ir a la carpeta de Controllers y dar clic derecho, elegir la opción de agregar y después elegir Controlador.

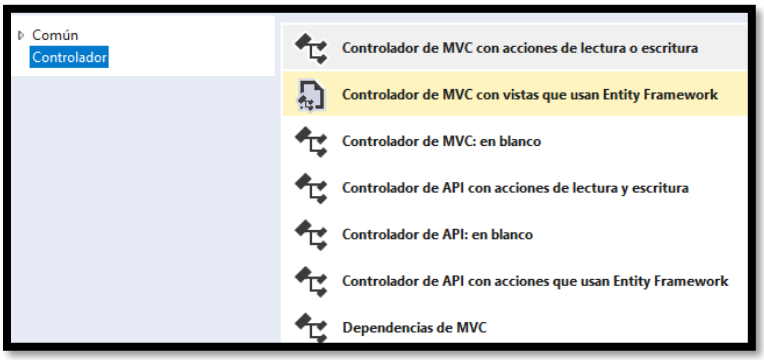
Ilustración 36 Pantalla creación de un Controlador



Representación de la creación de un controlador.

Posteriormente se elige el tipo de Controlador que se desea crear, en este caso se eligió controlador MVC con vistas que usan Entity Framework.

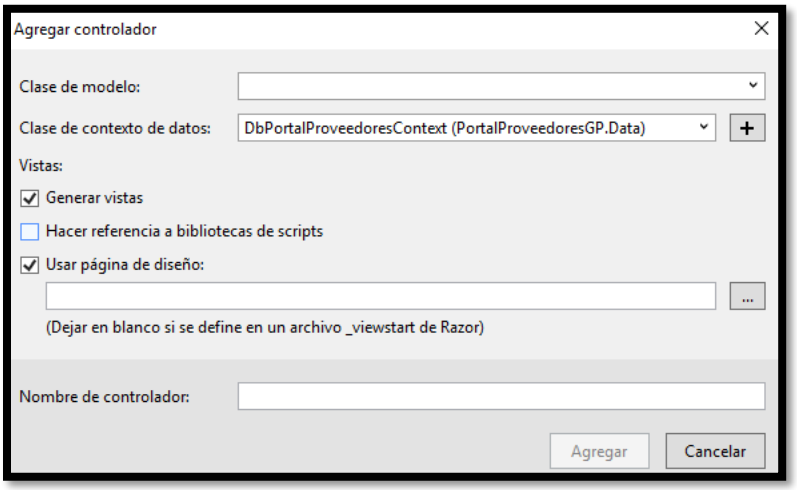
Ilustración 37 Pantalla elección de tipo de Controlador



Representación de elegir un tipo de controlador.

Se selecciona una clase de modelo, se elige un contexto, después se elige la opción de crear vistas y por último, se le asigna un nombre.

Ilustración 38 Pantalla definir propiedades del Controlador



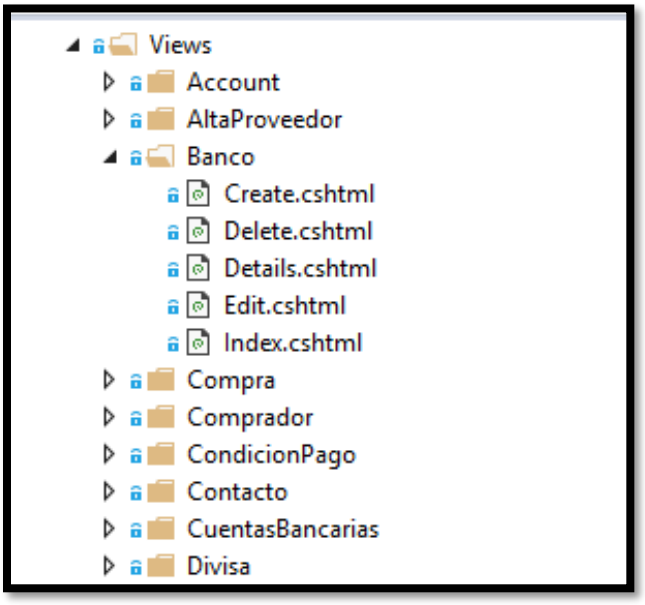
Representación de asignar propiedades al controlador.

En este apartado se puede observar como es la estructura de un controlador, existe un método que manda a llamar a la vista y otro que guarda los datos de esa vista.

```
public IActionResult Create()
      {
         return View();
      }

[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("ProveedorId,TipoPersonaId,RazonSocial
")] Proveedores proveedores)
      {
         if (ModelState.IsValid)
         { _context.Add(proveedores);
            await _context.SaveChangesAsync();
            return RedirectToAction("Index" });
         }
         return View(proveedores);
      }
```

Si se dirige a la carpeta Views y busca la carpeta que tenga el mismo nombre que el controlador, podrá observar que esa carpeta contiene 5 archivos los cuales son las vistas que el controlador generó.

Ilustración 39 Pantalla representación de vistas creadas por el Controlador

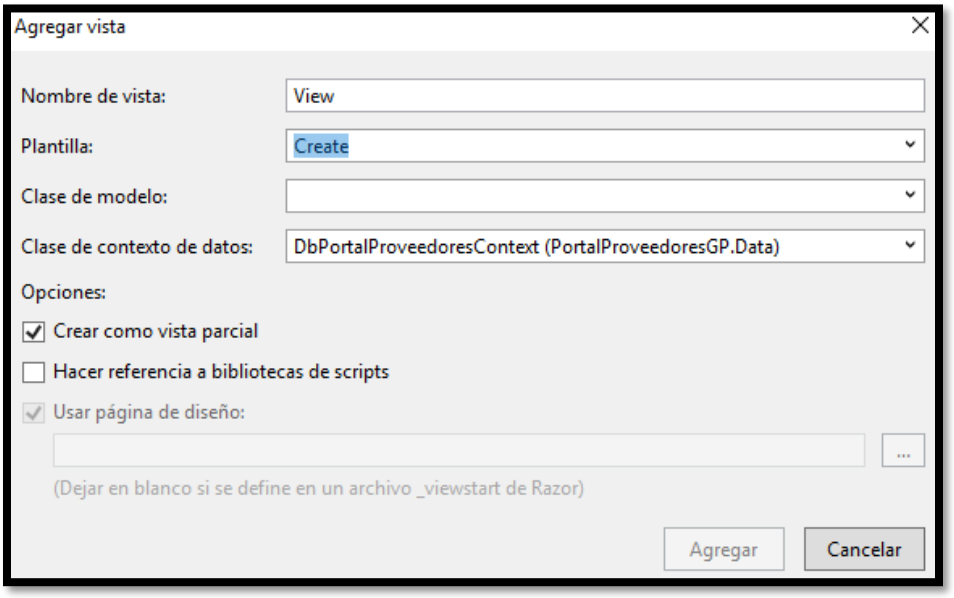


Representación de las vistas creadas por el controlador.

5.3.2.1.4 Generación de una Vista (View)

El siguiente paso después de crear un controlador es personalizarlo para visualizarlo. Esta fase se desarrolla con las vistas. Para ello se busca la carpeta en donde se desea crear la vista, se da clic derecho y aparecerá una ventana similar a la que se muestra a continuación. En la cual se debe de indicar el nombre de la vista, la plantilla es decir, la función que tendrá la vista (crear, editar, ver detalle, eliminar, index), el modelo al que pertenecerá, el contexto de la base de datos y otras características más. No todas las características mencionadas anteriormente son obligatorias al crear una vista, esto depende del tipo de vista que se desea crear.

Ilustración 40 Pantalla representación crear una Vista



Representación de agregar una vista.

De acuerdo a lo realizado anteriormente, se genera el siguiente código.

```
@using PortalProveedoresGP.Models
@using PortalProveedoresGP.Models.AccountViewModels
@using Microsoft.AspNetCore.Identity

@model RegisterViewModel
@{
    ViewData["Title"] = "Register";
}
```

Esta sección muestra código Razor dentro de la vista, la sintaxis es @{…} y permite ejecutar código C# para mejorar la funcionalidad de la vista tal como sea necesario.

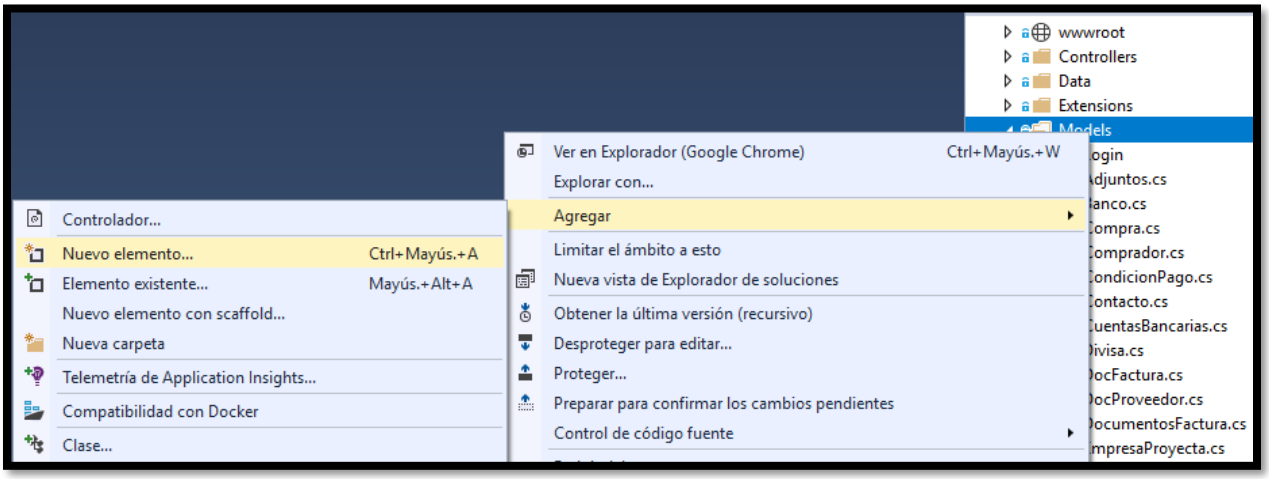
En esta sección se muestra código HTML y es el que se visualiza en el navegador web, se puede observar que contiene varios campos, los datos introducidos en los campos son los que se almacenarán en la base de datos.

```
<div class="col-lg-7 text-left bg-white align-self-center" style="height:100vh;">
<h4>Crea una nueva cuenta.</h4>
<div class="col-lg-12 align-self-end registroTop">
<form asp-route-returnUrl="@ViewData["ReturnUrl"]" method="post" class="w-100">
<div asp-validation-summary="All" class="text-danger"></div>
<div class="input-group">
<span class="input-group-addon" id="basic-mail"> Email </span>
<input type="email" asp-for="Email" class="form-control "
placeholder="nombre@correo.com" aria-label="correo" aria-describedby="basic-
email"  />
<span asp-validation-for="Email" class="text-danger"></span>
</div>
<br />
<div class="input-group">
<span class="input-group-addon" id="basic-pass">Pass </span>
<input asp-for="Password" class="form-control " placeholder="password" aria-
label="password" aria-describedby="basic-pass" />
<span asp-validation-for="Password" class="text-danger"></span>
</div>
<br />
<label asp-for="ConfirmPassword">Confirma el password</label>
<div class="input-group">
<span class="input-group-addon" id="basic-passB">Pass </span>
<input asp-for="ConfirmPassword" class="form-control"  placeholder="password"
aria-label="password" aria-describedby="basic-passB" />
<span asp-validation-for="ConfirmPassword" class="text-danger"></span>
</div>
<br />
<button type="submit" class="btn btn-primary">Registrar</button>
<br />
</form>
</div>
</div>
```

5.3.2.1.5 Generación de un Modelo (Model)

Un modelo es usado para el traspaso de información entre el controlador y la vista. El modelo se conecta a la base de datos y trae la información necesaria para poder gestionar datos desde MVC hacia la Base de Datos. Para crear un modelo, primero es necesario dirigirse a la carpeta Models, y con el botón derecho seleccionar Agregar, después elegir Nuevo elemento y de la lista que aparece seleccionar Clase.

Ilustración 41 Pantalla representación crear un Modelo



Representación de agregar un modelo.

A continuación, se muestra el código usado para la generación de un modelo, para ello es necesario declarar las librerías a usar, el tipo de dato del atributo, el nombre del atributo, las reglas de validación y las relaciones con otros modelos.

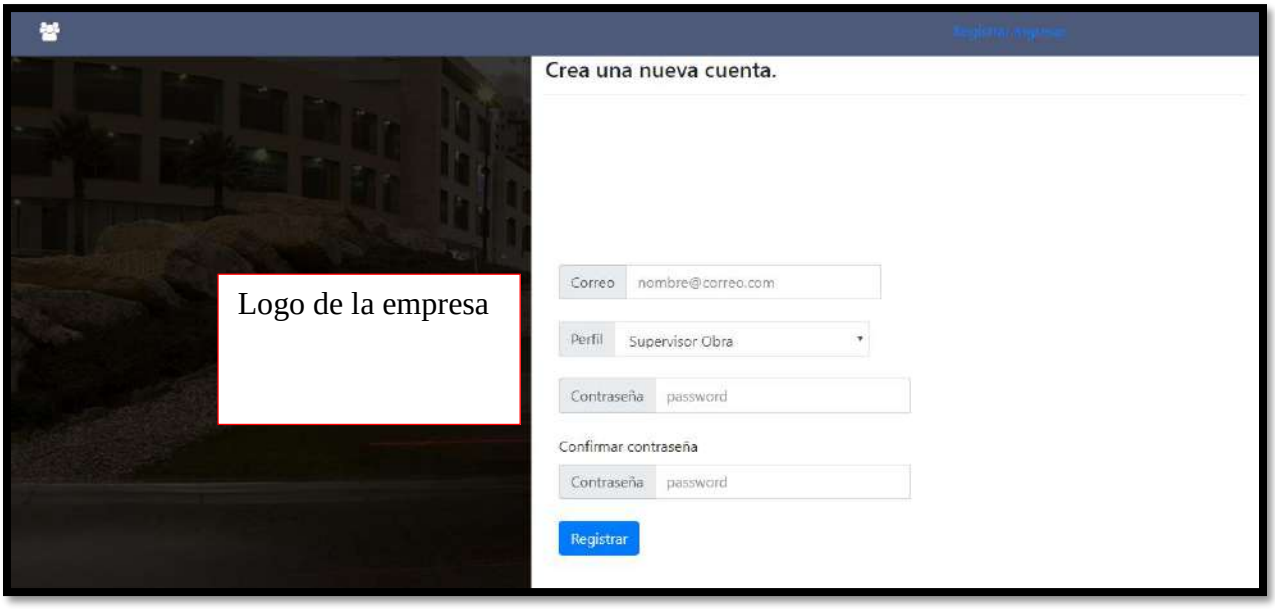
```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;

namespace PortalProveedoresGP.Models
{
    public partial class Proveedores
    {
        public Proveedores()
        {
            Compra = new HashSet<Compra>();
            Contacto = new HashSet<Contacto>();
            CuentasBancarias = new HashSet<CuentasBancarias>();
            ProveedoresMensajes = new HashSet<ProveedoresMensajes>();
        }

        public int ProveedorId { get; set; }
        [Required(ErrorMessage = "Falta Informacion")]
        public int? TipoPersonaId { get; set; }
        [Required(ErrorMessage ="Falta Informacion")]
        public string RazonSocial { get; set; }
        public string Nombres { get; set; }
        public string ApPaterno { get; set; }
        public string ApMaterno { get; set; }
        public int? TipoEmpresaId { get; set; }
        public string TipoOperacion { get; set; }
        public string Rfc { get; set; }
        public int? GrupoProveedoresId { get; set; }
        public string Calle { get; set; }
        public int? NoExt { get; set; }
        public int? NoInt { get; set; }
        public string Colonia { get; set; }
        public string DelMpio { get; set; }
        public string Pais { get; set; }
        public string EstadoRegion { get; set; }
        public int? Cp { get; set; }
        public string Telefono { get; set; }
        public int? ProveedorAxid { get; set; }
        public int? ProveedorOracleId { get; set; }

        public virtual ICollection<Compra> Compra { get; set; }
        public virtual ICollection<Contacto> Contacto { get; set; }
        public virtual ICollection<CuentasBancarias> CuentasBancarias { get; set; }
        public virtual GrupoProveedores GrupoProveedores { get; set; }
        public virtual TipoEmpresa TipoEmpresa { get; set; }
        public virtual TipoPersona TipoPersona { get; set; }
        public virtual ICollection<ProveedoresMensajes> ProveedoresMensajes { get; set; }

    }
}
```

A continuación, se detalla el desarrollo e implementación de las distintas pantallas que componen la interfaz de usuario.

5.3.2.1.5 Vista Crear Cuenta de Acceso

La misión de esta vista es crear un usuario para posteriormente éste pueda acceder al sistema, para ello es obligatorio proporcionar un correo electrónico una contraseña, por default el sistema le otorgara el rol de proveedor, pero el rol puede ser modificado por el administrador del sistema y con ello asignar el rol al usuario correspondiente.

Ilustración 42 Vista crear Cuenta de Acceso



Representación crear cuenta de acceso al sistema.

En la siguiente sección de código se muestra el modelo implementado para poder dar de alta un usuario. Esta clase contiene los elementos necesarios para trabajar y enlazar con la base de datos para poder realizar las acciones necesarias.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace PortalProveedoresGP.Models.AccountViewModels
{
    public class RegisterViewModel
    {
        [Required]
        [EmailAddress]
        [Display(Name = "Email")]
        public string Email { get; set; }

        [Required]
        [StringLength(100, ErrorMessage = "The {0} must be at least {2} and at max {1}
characters long.", MinimumLength = 6)]
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
        public string Password { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirm password")]
        [Compare("Password", ErrorMessage = "The password and confirmation password do
not match.")]
        public string ConfirmPassword { get; set; }
    }
}
```

Las siguientes dos secciones de código muestran los métodos usados para crear la vista y guardar los datos que el usuario proporcione.

```
[HttpGet]
[AllowAnonymous]
        [Authorize (Roles = "Admin")]//por role
        public IActionResult Register(string returnUrl = null)
        {
            ViewData["ReturnUrl"] = returnUrl;
            ViewData["RoleId"] = new SelectList(_context.Roles.ToList(), "Name",
"Name");
            return View();
        }
```

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task [IActionResult] Register(RegisterViewModel model, string returnUrl
= null)
   {
    ViewData["ReturnUrl"] = returnUrl;
    if (ModelState.IsValid)
      {
       var user = new ApplicationUser { UserName = model.Email, Email = model.Email
};
     var result = await _userManager.CreateAsync(user, model.Password);
     if (result.Succeeded)
      {
      _logger.LogInformation("User created a new account with password.");
       var code = await _userManager.GenerateEmailConfirmationTokenAsync(user);
       var callbackUrl = Url.EmailConfirmationLink(user.Id, code, Request.Scheme);
       await _emailSender.SendEmailConfirmationAsync(model.Email, callbackUrl);
       await AddUserToRole(user.Id, model.RoleId);
       await _signInManager.SignInAsync(user, isPersistent: false);
       _logger.LogInformation("User created a new account with password.");
       return RedirectToLocal(returnUrl);
     }
     AddErrors(result);
   }
    return View(model);
}
```

En este apartado se muestra el código generado en la vista crear cuenta de acceso.

```
@using PortalProveedores.Models
@using PortalProveedores.Models.AccountViewModels
@using Microsoft.AspNetCore.Identity
@model RegisterViewModel
@{
    ViewData["Title"] = "Register";
}
<div class="bgFixed bgProyecta1" style="height:88vh; overflow:hidden;">
    <div class="bgBlack50" style="width:100%; height:100vh">
        <div class="container col-lg-12" style="height:100vh;">
            <div class="row">
                @*columna 1*@
                <div class="col-lg-5 text-right align-self-center">
                    <div class="col-lg-12 align-self-end">
                      <img src="~/images/ic_grupo_proyecta_white.png" class="w-50" />
                        <br /><br />
                        <p class="text-white">
                            <b>Planeamos y Desarrollamos</b><br />
                            las mejores comunidades del país.
                        </p>
                    </div>
                </div>
               @*columna 2*@
```

```
<div class="col-lg-7 text-left bg-white align-self-center" style="height:100vh;">
<h4>Crea una nueva cuenta.</h4>
<hr />
<div class="col-lg-12 align-self-end registroTop">
<form asp-route-returnUrl="@ViewData["ReturnUrl"]" method="post" class="w-100">
<div asp-validation-summary="All" class="text-danger"></div>
<div class="input-group">
<span class="input-group-addon" id="basic-mail"> Correo </span>
<input type="email" asp-for="Email" class="form-control "
placeholder="nombre@correo.com" aria-label="correo" aria-describedby="basic-email"
/>
<span asp-validation-for="Email" class="text-danger"></span>
</div>
<br />
<div class="input-group">
<span class="input-group-addon"> Perfil </span>
<select asp-for="RoleId" class="form-control" asp-items="ViewBag.RoleId"></select>
<span asp-validation-for="RoleId" class="text-danger"></span>
</div>
<br />
<div class="input-group">
<span class="input-group-addon" id="basic-pass">Contraseña</span>
<input asp-for="Password" class="form-control " placeholder="password" aria-
label="password" aria-describedby="basic-pass" />
<span asp-validation-for="Password" class="text-danger"></span>
</div>
<br />
<label asp-for="ConfirmPassword">Confirmar contraseña</label>
<div class="input-group">
<span class="input-group-addon" id="basic-passB">Contraseña</span>
<input asp-for="ConfirmPassword" class="form-control"  placeholder="password" aria-
label="password" aria-describedby="basic-passB" />
<span asp-validation-for="ConfirmPassword" class="text-danger"></span>
</div>
<br />
<button type="submit" class="btn btn-primary">Registrar</button>
<br />
</form>
</div>
</div>
</div>
</div>
</div>
</div>
```

5.3.2.1.6 Vista Acceso al Sistema

El objetivo de esta vista es autenticar a un usuario existente en la base de datos, las vistas y acciones que el usuario podrá realizar dentro del sistema dependen del rol que se le haya proporcionado.

Ilustración 43 Vista Acceso al Sistema

Representación autenticación de un usuario.

En la siguiente sección de código se muestra el modelo implementado para que realice la comunicación con la base de datos y acepte o rechace un usuario que intenta acceder al sistema. Posteriormente se muestra los métodos implementado el controlador, los cuales realizan la comunicación entre el modelo y la vista que percibe el usuario.

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Threading.Tasks;

namespace PortalProveedores.Models.AccountViewModels
{
    public class LoginViewModel
    {
        [Required]
        [EmailAddress]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }

        [Display(Name = "Remember me?")]
        public bool RememberMe { get; set; }
    }
}
```

El siguiente método representa la acción que permite al usuario visualizar la vista y el método que se presenta en la otra sección de código realiza la comunicación con la base de datos para verificar si los datos son correctos o incorrectos.

```
[HttpPost]
[AllowAnonymous]
        public async Task<IActionResult> Login(string returnUrl = null)
        {
            // Clear the existing external cookie to ensure a clean login process
            await HttpContext.SignOutAsync(IdentityConstants.ExternalScheme);

            ViewData["ReturnUrl"] = returnUrl;
            return View();
        }
```

```
[HttpPost]
[AllowAnonymous]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Login(LoginViewModel model, string returnUrl = null)
       {
            ViewData["ReturnUrl"] = returnUrl;
            if (ModelState.IsValid)
            {
                 var result = await _signInManager.PasswordSignInAsync(model.Email,
              model.Password, model.RememberMe, lockoutOnFailure: false);
                if (result.Succeeded)
                {
                    _logger.LogInformation("User logged in.");
                    return RedirectToLocal(returnUrl);
                }
                if (result.RequiresTwoFactor)
                     { return RedirectToAction(nameof(LoginWith2fa), new { returnUrl,
              model.RememberMe });
                }
                if (result.IsLockedOut)
                {
                    _logger.LogWarning("User account locked out.");
                    return RedirectToAction(nameof(Lockout));
                }
                else
                {
                    ModelState.AddModelError(string.Empty, "Invalid login attempt.");
                    return View(model);
                }
            }

            return View(model);
        }
```

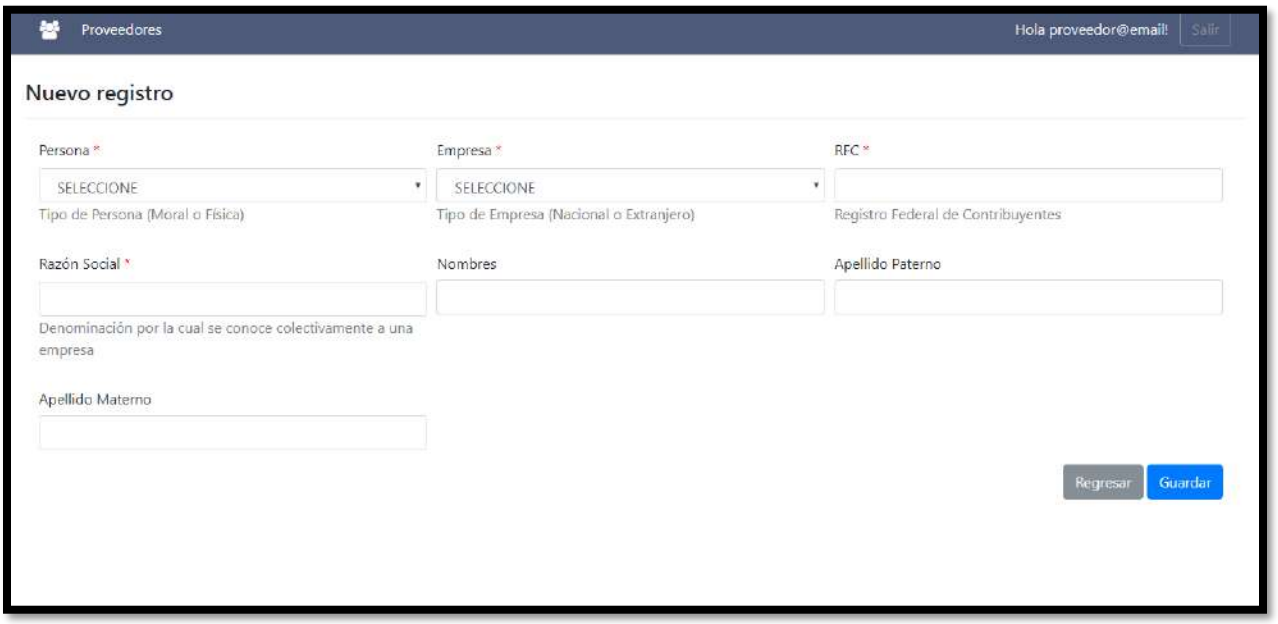
El código que se muestra a continuación, es el que se utilizó para generar los componentes de la vista. La vista se comunica con el controlador y este con el modelo para emitir resultados a la vista, por ejemplo, si el usuario o contraseña son incorrectos se muestra una etiqueta de error en la vista.

```
@model RegisterViewModel
@{
    ViewData["Title"] = "Register";
}

<div class="bgFixed bgProyecta1" style="height:88vh; overflow:hidden;">
<div class="bgBlack50" style="width:100%; height:100vh">
<div class="container col-lg-12" style="height:100vh;">
<div class="row">
<div class="col-lg-5 text-right align-self-center">
<div class="col-lg-12 align-self-end">
<img src="~/images/ic_grupo_proyecta_white.png" class="w-50" />
<p class="text-white">
<b>Planeamos y Desarrollamos</b><br />las mejores comunidades del país.
</p>
</div>
</div>
<div class="col-lg-7 text-left bg-white align-self-center" style="height:100vh;">
<h4>Crea una nueva cuenta.</h4>
<hr />
<div class="col-lg-12 align-self-end registroTop">
<form asp-route-returnUrl="@ViewData["ReturnUrl"]" method="post" class="w-100">
<div asp-validation-summary="All" class="text-danger"></div>
<div class="input-group">
<span class="input-group-addon" id="basic-mail"> Email </span>
<input type="email" asp-for="Email" class="form-control "
placeholder="nombre@correo.com" aria-label="correo" aria-describedby="basic-email"  />
<span asp-validation-for="Email" class="text-danger"></span>
</div>
<br />
<div class="input-group">
<span class="input-group-addon" id="basic-pass">Pass </span>
<input asp-for="Password" class="form-control " placeholder="password" aria-
label="password" aria-describedby="basic-pass" />
<span asp-validation-for="Password" class="text-danger"></span>
</div>
<br />
<label asp-for="ConfirmPassword">Confirma el password</label>
<div class="input-group">
<span class="input-group-addon" id="basic-passB">Pass </span>
<input asp-for="ConfirmPassword" class="form-control"  placeholder="password" aria-
label="password" aria-describedby="basic-passB" />
<span asp-validation-for="ConfirmPassword" class="text-danger"></span>
</div>
<br />
<button type="submit" class="btn btn-primary">Registrar</button>
<br />
</form>
</div>
</div>
</div>
</div>
</div>
</div>
```

5.3.2.1.7 Vista Crear Proveedor

La siguiente vista permite al usuario dar de alta un proveedor, para ello varios campos son obligatorios, es decir son necesarios para poder dar de alta a un proveedor.

Ilustración 44 Vista dar de alta un Proveedor



Representación crear proveedor.

El modelo que se necesitó para obtener todos los atributos del proveedor, es el siguiente note que los atributos de la clase proveedor no todos se utilizan en la vista anterior, esto se utilizarán posteriormente. Esta clase cuenta con varias relaciones a otras entidades, en la parte inferior podrá observar las relaciones.

```
    public partial class Proveedores
    {
        public Proveedores()
        {
            Compra = new HashSet<Compra>();
            Contacto = new HashSet<Contacto>();
            CuentasBancarias = new HashSet<CuentasBancarias>();
            ProveedoresMensajes = new HashSet<ProveedoresMensajes>();
        }
        public int ProveedorId { get; set; }
        [Required(ErrorMessage = "Falta Informacion")]
        public int? TipoPersonaId { get; set; }
        [Required(ErrorMessage ="Falta Informacion")]
        public string RazonSocial { get; set; }
        public string Nombres { get; set; }
        public string ApPaterno { get; set; }
        public string ApMaterno { get; set; }
        public int? TipoEmpresaId { get; set; }
        public string TipoOperacion { get; set; }
        public string Rfc { get; set; }
        public int? GrupoProveedoresId { get; set; }
        public string Calle { get; set; }
        public int? NoExt { get; set; }
        public int? NoInt { get; set; }
        public string Colonia { get; set; }
        public string DelMpio { get; set; }
        public string Pais { get; set; }
        public string EstadoRegion { get; set; }
        public int? Cp { get; set; }
        public string Telefono { get; set; }
        public int? ProveedorAxid { get; set; }
        public int? ProveedorOracleId { get; set; }
        public virtual ICollection<Compra> Compra { get; set; }
        public virtual ICollection<Contacto> Contacto { get; set; }
        public virtual ICollection<CuentasBancarias> CuentasBancarias { get; set; }
        public virtual GrupoProveedores GrupoProveedores { get; set; }
        public virtual TipoEmpresa TipoEmpresa { get; set; }
        public virtual TipoPersona TipoPersona { get; set; }
        public virtual ICollection<ProveedoresMensajes> ProveedoresMensajes { get;
set; }

    }
```

Los métodos del controlador necesarios para la vista crear se muestran a continuación, el primer método es necesario para mostrar la vista al usuario y el segundo método es para enviar o guardar los datos que el usuario proporcionó.

```
  public IActionResult Create()
       {
            ViewData["GrupoProveedoresId"] = new
SelectList(_context.GrupoProveedores, "GrupoProveedoresId", "Descripcion");
            ViewData["TipoEmpresaId"] = new SelectList(_context.TipoEmpresa,
"TipoEmpresaId", "Descripcion");
            ViewData["TipoPersonaId"] = new SelectList(_context.TipoPersona,
"TipoPersonaId", "Descripcion");
            ViewData["FormaPagoId"] = new SelectList(_context.FormaPago,
"FormaPagoId", "Descripcion");
            ViewData["BancoId"] = new SelectList(_context.Banco, "BancoId",
"Descripcion");
            ViewData["DivisaId"] = new SelectList(_context.Divisa, "DivisaId",
"Descripcion");
            ViewData["CondicionPagoId"] = new SelectList(_context.CondicionPago,
"CondicionPagoId", "Descripcion");
            return View();
        }

[HttpPost]
ValidateAntiForgeryToken]
public async Task<IActionResult>
Create([Bind("ProveedorId,TipoPersonaId,RazonSocial,Nombres,ApPaterno,ApMaterno,TipoE
mpresaId,Rfc,TipoOperacion,GrupoProveedoresId,Calle,NoExt,NoInt,Colonia,DelMpio,Pais,
EstadoRegion,Cp,Telefono,ProveedorAxid,ProveedorOracleId")] Proveedores proveedores)
        {
            if (ModelState.IsValid)
            {
                _context.Add(proveedores);
                await _context.SaveChangesAsync();
                int ProveedorId= proveedores.ProveedorId;
                return RedirectToAction("Edit", "AltaProveedor", new { id =
ProveedorId });

            }
            ViewData["GrupoProveedoresId"] = new
SelectList(_context.GrupoProveedores, "GrupoProveedoresId", "Descripcion",
proveedores.GrupoProveedoresId);
            ViewData["TipoEmpresaId"] = new SelectList(_context.TipoEmpresa,
"TipoEmpresaId", "Descripcion", proveedores.TipoEmpresaId);
            ViewData["TipoPersonaId"] = new SelectList(_context.TipoPersona,
"TipoPersonaId", "Descripcion", proveedores.TipoPersonaId);
            ViewData["FormaPagoId"] = new
SelectList(_context.FormaPago,"FormaPagoId","Descripcion");

            return View(proveedores);
        }
```

El código necesario para el funcionamiento de la vista es el que a continuación se presenta. En la vista se incluyó código Razor, HTML y JavaScript con la finalidad de que la vista se comporte como el cliente la necesita.

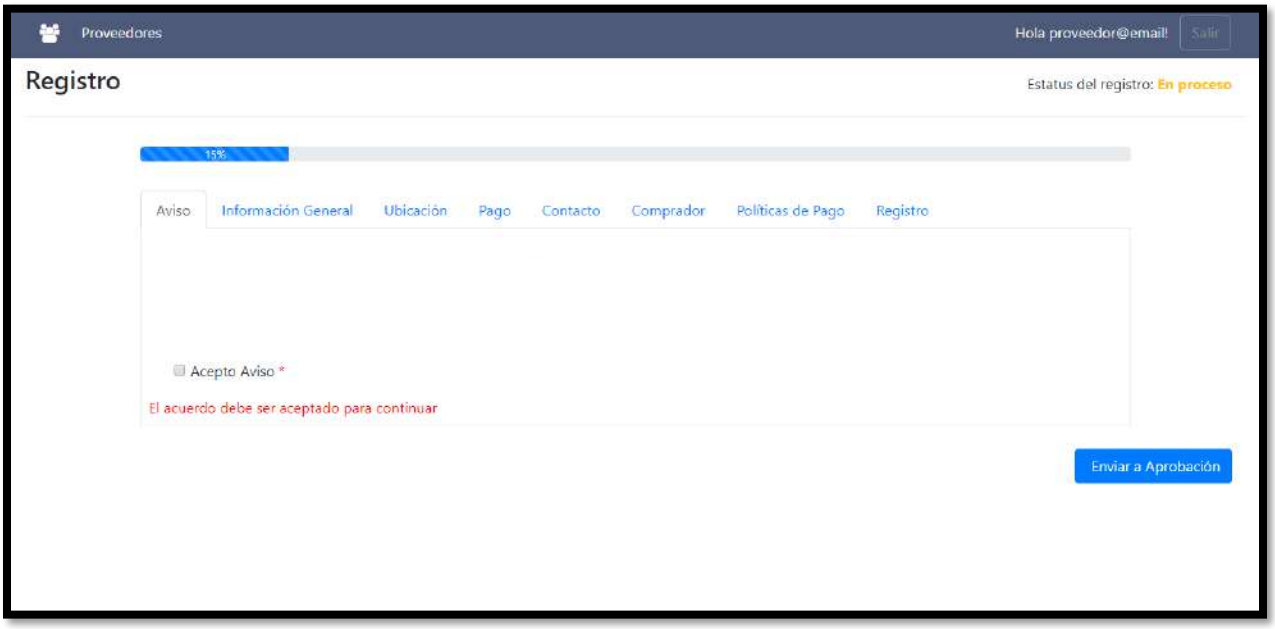
```
@model PortalProveedoresGP.Models.Proveedores
<form asp-action="Create">
<div class="form-row col-lg-12 ">
<div class="form-group col-lg-4 col-md-12 col-sm-12">
<label asp-for="TipoPersona" class="col-form-label">Persona</label>
<select required asp-for="TipoPersonaId" onchange="mostrarInput(this.value)"
id="Persona" class="form-control col-lg-12 col-md-12 col-sm-12 selectpicker" asp-
items="ViewBag.TipoPersonaId">
<option value="" selected disabled>SELECCIONE</option>
</select>
</div>
<div class="form-group col-lg-4 col-md-12 col-sm-12">
<label asp-for="TipoEmpresaId" class="col-form-label">Empresa</label>
<select required asp-for="TipoEmpresaId" onchange="mostrarRFC(this.value)"
id="Empresa" class="form-control col-lg-12 col-md-12 col-sm-12" asp-
items="ViewBag.TipoEmpresaId">
<option value="" selected disabled>SELECCIONE</option>
</select>
</div>
<div class="form-group col-lg-4 col-md-12 col-sm-12">
<label asp-for="Rfc" class="col-form-label"> RFC</label>
<input asp-for="Rfc" onkeyup="this.value=this.value.toUpperCase()" id="rfc"
class="form-control col-lg-12 col-md-12 col-sm-12" oninput="validarRFC(this)"  />
</div></div>
<div class="form-row col-lg-12 ">
<div class="form-group col-lg-4 col-md-12 col-sm-12">
<label asp-for="RazonSocial" class="col-form-label"> Razon Social</label>
<input asp-for="RazonSocial" onkeyup="this.value=this.value.toUpperCase()" required
class="form-control col-lg-12 col-md-12 col-sm-12" />
</div><div class="form-group col-lg-4 col-md-12 col-sm-12" id="nombre">
<label asp-for="Nombres" class="col-form-label"> Nombres</label>
<input asp-for="Nombres" onkeyup="this.value=this.value.toUpperCase()" id="nombres"
onkeypress="return soloLetras(event)" class="form-control col-lg-12 col-md-12 col-sm-
12" /></div>
<div class="form-group col-lg-4 col-md-12 col-sm-12" id="app">
<label asp-for="ApPaterno" class="col-form-label">Apellido Paterno</label>
<input asp-for="ApPaterno" onkeyup="this.value=this.value.toUpperCase()"
id="appaterno" onkeypress="return soloLetras(event)" class="form-control col-lg-12
col-md-12 col-sm-12" /></div></div>
<div class="form-row col-lg-12 " id="apm">
<div class="form-group col-lg-4 col-md-12 col-sm-12">
<label asp-for="ApMaterno" class="col-form-label">Apellido Materno</label>
<input asp-for="ApMaterno" onkeyup="this.value=this.value.toUpperCase()"
id="apmaterno" onkeypress="return soloLetras(event)" class="form-control col-lg-12
col-md-12 col-sm-12" />
<span asp-validation-for="ApMaterno" class="text-danger"></span>
</div>
</div>
<input type="button" id="Cancelar" value="Regresar" class="btn btn-secondary" />
<input type="submit" value="Guardar" onclick="return validar()" class="btn btn-
primary" /></div></form>
```

```
        $('#Cancelar').on('click', function () {
            var url = "/Proveedores/Index";
            window.location.href = url;
        });
        //funcion para mostrar y ocultar campos dependiendo del tipo de persona 1 =
Moral, 2 = Fisica
        function mostrarInput(valor) {
            if (valor == 1) {
                $("#nombre").hide();
                $("#app").hide();
                $("#apm").hide();
                $("#rfc").attr("readonly", "readonly");
                $("#rfc").addClass("readOnly");
                $('#Empresa').val('');
                $("#nombres").removeAttr("required");
                $("#appaterno").removeAttr("required");
                $("#apmaterno").removeAttr("required");
                $("#nombres").val("");
                $("#appaterno").val("");
                $("#apmaterno").val("");
            } else {
                $('#Empresa').val('');
                $("#nombre").show();
                $("#app").show();
                $("#apm").show();
                $("#rfc").removeAttr("readonly");
                $("#rfc").removeClass("readOnly");
                $("#nombres").attr("required", "required");
                $("#appaterno").attr("required", "required");
                $("#apmaterno").attr("required", "required");
            }
        }
        //funcion para permitir insertar RFC dependiendo del tipo de empresa 1 =
Nacional, 2 = Extranjera
        function mostrarRFC(valor) {
            if (valor == 1) {
                $("#rfc").removeAttr("readonly");
                $("#rfc").removeClass("readOnly");
                $("#rfc").attr("required", "required");

            } else {
                $("#rfc").val("");
                $("#msjrfc").hide();
                $("#rfc").attr("readonly", "readonly");
                $("#rfc").addClass("readOnly");
            }
        } //funcion validar solo letras
        function soloLetras(e) {
            tecla = (document.all) ? e.keyCode : e.which;
            if (tecla == 8) return true;
            patron = /[a-zA-
Z\s\u00D1\u00F1\u00e1\u00e9\u00ed\u00f3\u00fa\u00c1\u00c9\u00cd\u00d3\u00da]/;
            te = String.fromCharCode(tecla);
            return patron.test(te); }
```

5.3.2.1.8 Vista proporcionar información del proveedor

Esta vista tiene la finalidad de recabar datos referentes al proveedor, entre estos se incluyen los datos de ubicación, datos de pago, datos de contacto, documentos, compradores, aceptar políticas, entre otros.

Ilustración 45 Vista general para recabar datos del proveedor



Representación de la vista para recabar información referente al proveedor.

Como ya se mencionó anteriormente en esta vista el usuario debe proporcionar toda la información relacionada con el proveedor, para que esto pueda ser posible se optó por usar una vista parcial, de presentar varias vistas con diferentes modelos en una sola vista.

En las siguientes secciones de código se mostrarán algunos de los modelos utilizados, cabe mencionar que no se proporcionará todo el código por motivos de privacidad y también por el tamaño de los mismos.

Primeramente, se dará a conocer el modelo para la gestión de los datos de pago.

```
    public partial class CuentasBancarias
    {
        public int CuentasBancariasId { get; set; }
        public int? FormaPagoId { get; set; }
        public int? CondicionPagoId { get; set; }
        public int? BancoId { get; set; }
        public string Sucursal { get; set; }
        public string NumeroCuenta { get; set; }
        public string ClabeInterbancaria { get; set; }
        public int? DivisaId { get; set; }
        public string Titular { get; set; }
        public int? ProveedorId { get; set; }
        public virtual Banco Banco { get; set; }
        public virtual CondicionPago CondicionPago { get; set; }
        public virtual Divisa Divisa { get; set; }
        public virtual FormaPago FormaPago { get; set; }
        public virtual Proveedores  Proveedores { get; set; }
    }
}
```

En esta sección se muestra el modelo para los contactos.

```
    public partial class Contacto
    {
        public int ContactoId { get; set; }
        public string Nombre { get; set; }
        public string ApPaterno { get; set; }
        public string ApMaterno { get; set; }
        public string TelefonoOficina { get; set; }
        public int? Ext { get; set; }
        public string CorreoElectronico { get; set; }
        public int? NumeroFax { get; set; }
        public string TelefonoMovil { get; set; }
        public string TelefonoParticular { get; set; }
        public int? TipoContactoId { get; set; }
        public int? ProveedorId { get; set; }
        public int? ProveedorAxid { get; set; }

        public virtual Proveedores Proveedor { get; set; }
        public virtual TipoContacto TipoContacto { get; set; }
    }
}
```

Por último, se mostrará el modelo para la gestión de los documentos, para este caso se usó un ViewModel, con la finalidad usar un mismo modelo para todas aquellas vistas en donde sea necesarios manipular documentos o archivos.

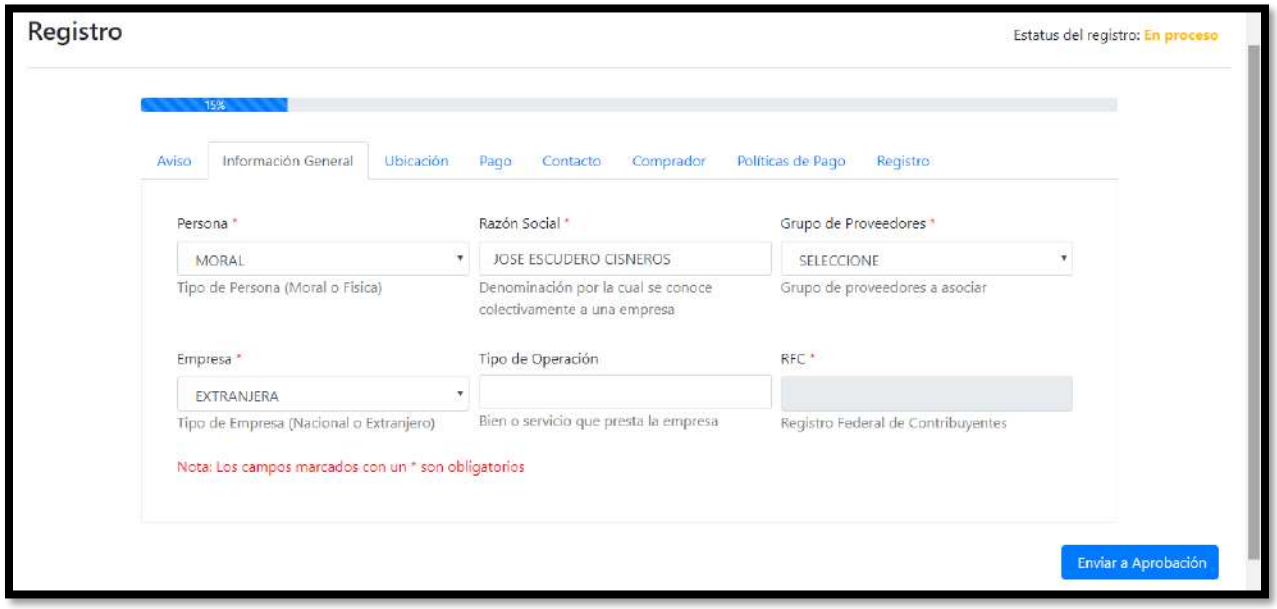
```
public class Adjuntos
    {
        public int AdjuntosId { get; set; }
        [StringLength(255)]
        public string FileName { get; set; }
        [StringLength(100)]
        public string ContentType { get; set; }
        public byte[] Content { get; set; }
        public string FileType { get; set; }
        public bool DefaultImg { get; set; }
    }
```

El código anterior representa a una clase la cual contiene los atributos necesarios de un documento, esta se usó en conjunto con la siguiente clase, la cual proporciona otros atributos para diferenciar a los documentos.

```
    public class DocProveedor:Adjuntos
    {
        public int ProveedorId { get; set; }
        public Proveedores Proveedores { get; set; }

        public int TipoDocumentoId { get; set; }
        public virtual TipoDocumento TipoDocumento { get; set; }
    }
```

En los siguientes apartados se darán a conocer de forma parcial las vistas más importantes, los métodos y partes de código necesarios para la funcionalidad del sistema.

Ilustración 46 Vista Datos Generales



Representación de la vista para recabar datos generales del proveedor.

Para visualizar la vista anterior y guardar los datos proporcionados por el usuario, son necesarios implementar dos métodos en el controlador, uno que haga una petición para ver el contenido y otro que envié los datos para que sean almacenados en la base de datos.

```
public async Task<IActionResult> DatosGenerales(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            ViewData["ProveedorId"] = id;
            var proveedores = await _context.Proveedores.SingleOrDefaultAsync(m =>
m.ProveedorId == id);
            if (proveedores == null){return NotFound();
            }
            ViewData["GrupoProveedoresId"] = new SelectList(_context.GrupoProveedores,
"GrupoProveedoresId", "Descripcion", proveedores.GrupoProveedoresId);
            ViewData["TipoEmpresaId"] = new SelectList(_context.TipoEmpresa,
"TipoEmpresaId", "Descripcion", proveedores.TipoEmpresaId);
            ViewData["TipoPersonaId"] = new SelectList(_context.TipoPersona,
"TipoPersonaId", "Descripcion", proveedores.TipoPersonaId);
            return PartialView(proveedores);
        }
```

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DatosGenerales(int id,
[Bind("ProveedorId,TipoPersonaId,RazonSocial,Nombres,ApPaterno,ApMaterno,TipoEmpresaI
d,TipoOperacion,Rfc,GrupoProveedoresId,Calle,NoExt,NoInt,Colonia,DelMpio,Pais,EstadoR
egion,Cp,Telefono,ProveedorAxid,ProveedorOracleId")] Proveedores proveedores)
       {
            if (id != proveedores.ProveedorId)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(proveedores);
                    await _context.SaveChangesAsync();

                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!ProveedoresExists(proveedores.ProveedorId))
                    {
                        return NotFound();
                    }
                    else
                    {

                        throw;
                    }
                }
                int ProveedorId = proveedores.ProveedorId;
                return RedirectToAction("Edit", "AltaProveedor", new { id =
ProveedorId });
            }
            else
            {
                ViewBag.Message = string.Format("Falta Informacion");
            }
            return PartialView(proveedores);

        }
```

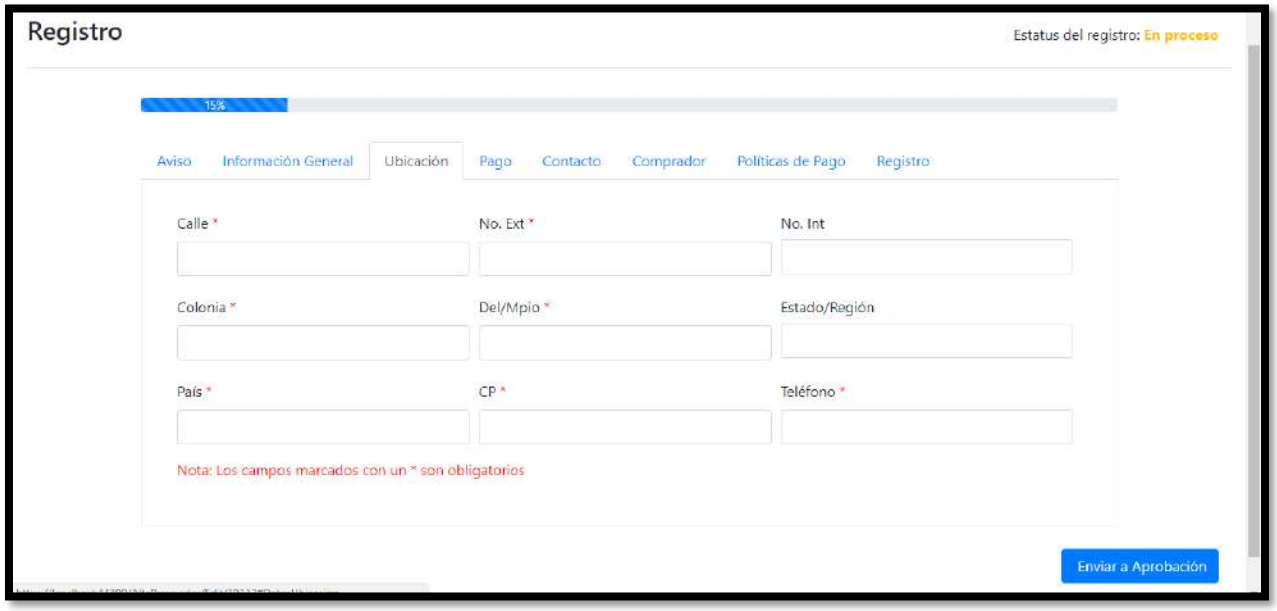
El código que se usó para esta vista es el siguiente, en este apartado también se emplea código JavaScript para complementar la funcionalidad su funcionalidad.

```
@model PortalProveedoresGP.Models.Proveedores
<div class="col-md-12 paddingSuperior2">
<form asp-action="DatosGenerales/@ViewData["ProveedorId"]">
<div class="form-horizontal">
<div asp-validation-summary="ModelOnly" class="text-danger"></div>
<input type="hidden" asp-for="ProveedorId" />
<div class="form-row col-lg-12 ">
<div class="form-group col-lg-4 col-md-12">
<label asp-for="TipoPersonaId" class="col-form-label">Persona</label>
<select asp-for="TipoPersonaId" onchange="mostrarInput(this.value)" id="Persona" asp-
items="ViewBag.TipoPersonaId" class="form-control col-lg-12 col-md-12 col-sm-12">
<option value="" selected disabled>SELECCIONE</option>
</select>
<span asp-validation-for="TipoPersonaId" class="text-danger"></span>
</div>
<div class="form-group col-lg-4 col-md-12">
<label asp-for="RazonSocial" class="col-form-label">Razon Social</label>
<input asp-for="RazonSocial" id="RazonS"
onkeyup="this.value=this.value.toUpperCase()" required class="form-control col-lg-12
col-md-12 col-sm-12" />
<span asp-validation-for="RazonSocial" class="text-danger"></span>
</div>
<div class="form-group col-lg-4 col-md-12">
<label asp-for="GrupoProveedoresId" class="col-form-label">Grupo de
Proveedores</label>
<select asp-for="GrupoProveedoresId" id="GrupoProv" class="form-control col-lg-12
col-md-12 col-sm-12" asp-items="ViewBag.GrupoProveedoresId" required>
<option value="" selected disabled>SELECCIONE</option>
</select>
<span asp-validation-for="GrupoProveedoresId" class="text-danger"></span>
</div>
</div>
<div class="form-row col-lg-12" id="nombres">
<div class="form-group col-lg-4 col-md-12">
<label asp-for="Nombres" id="nombreslabel" class="col-form-label"></label>
<input asp-for="Nombres" id="nombre" onkeyup="this.value=this.value.toUpperCase()"
onkeypress="return soloLetras(event)" class="form-control col-lg-12 col-md-12 col-sm-
12" />
<span asp-validation-for="Nombres" class="text-danger"></span>
</div>
<div class="form-group col-lg-4 col-md-12">
<label asp-for="ApPaterno" class="col-form-label">Apellido Paterno</label>
<input asp-for="ApPaterno" id="appaterno"
onkeyup="this.value=this.value.toUpperCase()" onkeypress="return soloLetras(event)"
class="form-control col-lg-12 col-md-12 col-sm-12" />
<span asp-validation-for="ApPaterno" class="text-danger"></span>
</div>
<div class="form-group col-lg-4 col-md-12">
<label asp-for="ApMaterno" class="col-form-label">Apellido Materno</label>
<input asp-for="ApMaterno" id="apmaterno"
onkeyup="this.value=this.value.toUpperCase()" onkeypress="return soloLetras(event)"
class="form-control col-lg-12 col-md-12 col-sm-12" />
<span asp-validation-for="ApMaterno" class="text-danger"></span>
</div>
</div></form>
```

Parte del código JavaScript utilizado es el que a continuación se muestra.

```
<script type="text/javascript">
        //validar tipo de emprsa al cargar la pagina
        var idempresa = $("#Empresa").val();
        if (idempresa == 1) {
            $("#rfc").removeAttr("readonly");
            $("#rfc").removeClass("readOnly");
        } else {
            $("#rfc").attr("readonly", "readonly");
            $("#rfc").addClass("readOnly");
        }

    });
    //funcion para validar rfc
    var rfcCorrecto;
    function rfcValido(rfc, aceptarGenerico = true) {
        const re = /^([A-ZÑ&]{3,4}) ?(?:- ?)?(\d{2}(?:0[1-9]|1[0-2])(?:0[1-
9]|[12]\d|3[01])) ?(?:- ?)?([A-Z\d]{2})([A\d])$/;
        var validado = rfc.match(re);
        if (!validado)  //Coincide con el formato general del regex?
            return false;
        //Separar el dígito verificador del resto del RFC
        const digitoVerificador = validado.pop(),
            rfcSinDigito = validado.slice(1).join(''),
            len = rfcSinDigito.length,
            //Obtener el digito esperado
            diccionario = "0123456789ABCDEFGHIJKLMN&OPQRSTUVWXYZ Ñ",
            indice = len + 1;
        var suma,
            digitoEsperado;
        if (len == 12) suma = 0
        else suma = 481; //Ajuste para persona moral

        for (var i = 0; i < len; i++)
            suma += diccionario.indexOf(rfcSinDigito.charAt(i)) * (indice - i);
        digitoEsperado = 11 - suma % 11;
        if (digitoEsperado == 11) digitoEsperado = 0;
        else if (digitoEsperado == 10) digitoEsperado = "A";

        //El dígito verificador coincide con el esperado?
        // o es un RFC Genérico (ventas a público general)?
        if ((digitoVerificador != digitoEsperado)
            && (!aceptarGenerico || rfcSinDigito + digitoVerificador !=
"XAXX010101000"))
            return false;
        else if (!aceptarGenerico && rfcSinDigito + digitoVerificador ==
"XEXX010101000")
            return false;
        return rfcSinDigito + digitoVerificador;
    }

</script>
```

Ilustración 47 Vista Datos Ubicación



Representación de la vista para recabar datos de ubicación del proveedor.

Métodos del controlador que se utilizaron, primero el método para presentar a la vista y después le método para enviar lo datos a guardar.

```
public async Task<IActionResult> DatosUbicacion(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            ViewData["ProveedorId"] = id;
            var proveedores = await _context.Proveedores.SingleOrDefaultAsync(m =>
m.ProveedorId == id);
            if (proveedores == null)
            {
                return NotFound();
            }
            ViewData["GrupoProveedoresId"] = new SelectList(_context.GrupoProveedores,
"GrupoProveedoresId", "Descripcion", proveedores.GrupoProveedoresId);
            ViewData["TipoEmpresaId"] = new SelectList(_context.TipoEmpresa,
"TipoEmpresaId", "Descripcion", proveedores.TipoEmpresaId);
            ViewData["TipoPersonaId"] = new SelectList(_context.TipoPersona,
"TipoPersonaId", "Descripcion", proveedores.TipoPersonaId);

            return PartialView(proveedores);
        }
```

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> DatosUbicacion(int id,
[Bind("ProveedorId,TipoPersonaId,RazonSocial,Nombres,ApPaterno,ApMaterno,TipoEmpresaId
,TipoOperacion,Rfc,GrupoProveedoresId,Calle,NoExt,NoInt,Colonia,DelMpio,Pais,EstadoReg
ion,Cp,Telefono,ProveedorAxid,ProveedorOracleId")] Proveedores proveedores)
        {
            if (id != proveedores.ProveedorId)
            {
                return NotFound();
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(proveedores);
                    await _context.SaveChangesAsync();

                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!ProveedoresExists(proveedores.ProveedorId))
                    {
                        return NotFound();
                    }
                    else
                    {

                        throw;
                    }
                }
                int ProveedorId = proveedores.ProveedorId;
                return RedirectToAction("Edit", "AltaProveedor", new { id =
ProveedorId});
            }
            else
            {
                ViewBag.Message = string.Format("Falta Informacion");
            }

            ViewData["GrupoProveedoresId"] = new SelectList(_context.GrupoProveedores,
"GrupoProveedoresId", "Descripcion", proveedores.GrupoProveedoresId);
            ViewData["TipoEmpresaId"] = new SelectList(_context.TipoEmpresa,
"TipoEmpresaId", "Descripcion", proveedores.TipoEmpresaId);
            ViewData["TipoPersonaId"] = new SelectList(_context.TipoPersona,
"TipoPersonaId", "Descripcion", proveedores.TipoPersonaId);
            return PartialView(proveedores);

        }
```

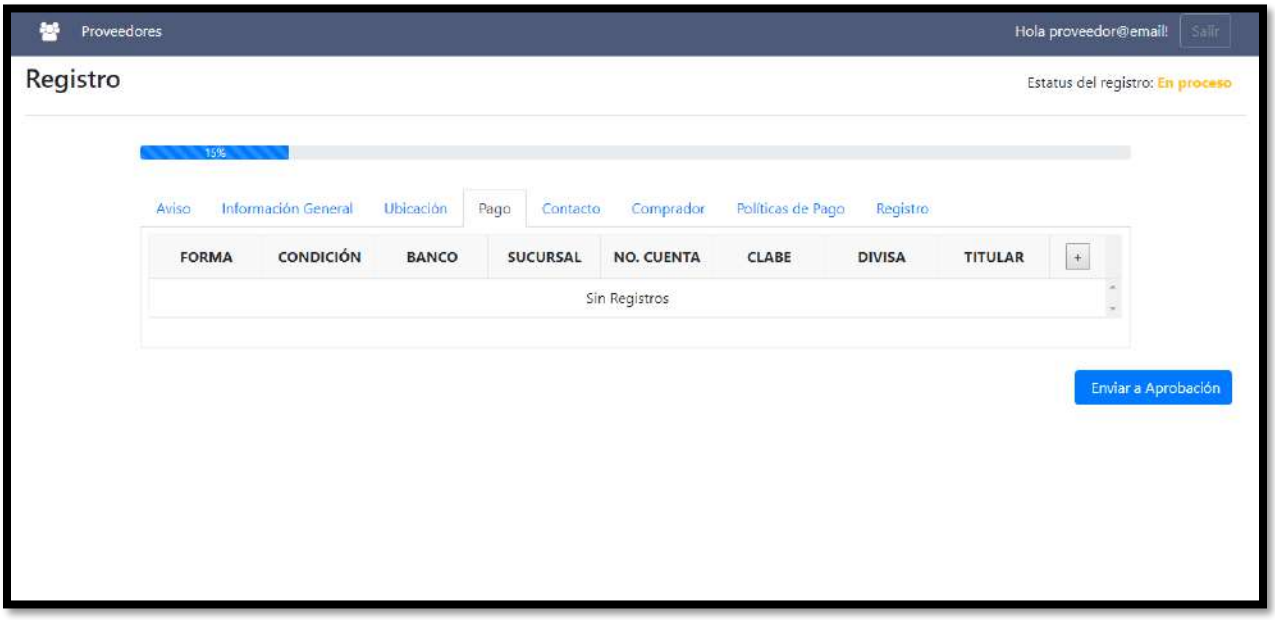
Código implementado en la vista de datos de ubicación.

```
@model PortalProveedoresGP.Models.Proveedores
<form asp-action="DatosUbicacion/@ViewData["ProveedorId"]">
<div class="form-horizontal">
<div asp-validation-summary="ModelOnly" class="text-danger"></div>
<input type="hidden" asp-for="ProveedorId" />
<div class="form-group col-lg-4 col-md-12 col-sm-12">
<label asp-for="Calle" id="calles" class="col-form-label"></label>
<input asp-for="Calle" onkeyup="this.value=this.value.toUpperCase()"
onkeypress="return direccion(event)" required class="form-control col-lg-12 col-md-12
col-sm-12" />
</div>
<div class="form-group col-lg-4 col-md-12 col-sm-12">
<label asp-for="NoExt" class="col-form-label">No. Ext</label>
<input asp-for="NoExt" type="tel" onKeyPress="return soloNumeros(event)" required
class="form-control col-lg-12 col-md-12 col-sm-12" />
</div>
<div class="form-group col-lg-4 col-md-12 col-sm-12">
<label asp-for="NoInt" class="col-form-label">No. Int</label>
<input asp-for="NoInt" type="tel" onKeyPress="return soloNumeros(event)" class="form-
control col-lg-12 col-md-12 col-sm-12" />
<div class="form-group col-lg-4 col-md-12 col-sm-12">
<label asp-for="Colonia" class="col-form-label"></label>
<input asp-for="Colonia" onkeyup="this.value=this.value.toUpperCase()"
onkeypress="return direccion(event)" required class="form-control col-lg-12 col-md-12
col-sm-12" />
</div>
<div class="form-group col-lg-4 col-md-12 col-sm-12">
<label asp-for="DelMpio" class="col-form-label">Del/Mpio</label>
<input asp-for="DelMpio" onkeyup="this.value=this.value.toUpperCase()"
onkeypress="return soloLetras(event)" required class="form-control col-lg-12 col-md-12
col-sm-12" />
</div>
<div class="form-group col-lg-4 col-md-12 col-sm-12">
<label asp-for="Pais" class="col-form-label"></label>
<input asp-for="Pais" onkeyup="this.value=this.value.toUpperCase()" onkeypress="return
soloLetras(event)" required class="form-control col-lg-12 col-md-12 col-sm-12" />
<div class="form-row col-lg-12 ">
<div class="form-group col-lg-4 col-md-12 col-sm-12">
<label asp-for="EstadoRegion" class="col-form-label">Estado/Region</label>
<input asp-for="EstadoRegion" onkeyup="this.value=this.value.toUpperCase()"
onkeypress="return soloLetras(event)" class="form-control col-lg-12 col-md-12 col-sm-
12" />
</div><div class="form-group col-lg-4 col-md-12 col-sm-12">
<label asp-for="Cp" class="col-form-label">CP</label>
<input asp-for="Cp" type="tel" maxlength="5" id="cp" onKeyPress="return
soloNumeros(event)" onkeyup="validacp()" required class="form-control col-lg-12 col-
md-12 col-sm-12" /></div>
<div class="form-group col-lg-4 col-md-12 col-sm-12">
<label asp-for="Telefono" class="col-form-label"></label>
<input type="tel" asp-for="Telefono" maxlength="13" id="telefono" onKeyPress="return
soloNumeros(event)" onkeyup="validatel()" required class="form-control col-lg-12 col-
md-12 col-sm-12" />
<input type="submit" onclick="return validacion()" value="Guardar" class="btn btn-
primary" /></form>
```

Código JavaScript utilizado para la vista de datos de ubicación.

```
<script type="text/javascript">
        function soloLetras(e) {
            tecla = (document.all) ? e.keyCode : e.which;
            if (tecla == 8) return true;
            patron = /[a-zA-
Z\s\u00D1\u00F1\u00e1\u00e9\u00ed\u00f3\u00fa\u00c1\u00c9\u00cd\u00d3\u00da]/;
            te = String.fromCharCode(tecla);
            return patron.test(te);
        }
        function direccion(e) {
            tecla = (document.all) ? e.keyCode : e.which;
            if (tecla == 8) return true;
            patron = /[a-zA-Z\0-
9\s\u00D1\u00F1\u00e1\u00e9\u00ed\u00f3\u00fa\u00c1\u00c9\u00cd\u00d3\u00da]/;
            te = String.fromCharCode(tecla);
            return patron.test(te);
        }
        function soloNumeros(e) {
            var key = window.Event ? e.which : e.keyCode
            return ((key >= 48 && key <= 57) || (key == 8))
        }

    //valida cp al presionar tecla
    function validacp() {
        var valorcp = document.getElementById("cp").value;
        //validar CP
        if (!validCpMx(valorcp)) { //NO VALIDO
            $("#msjcp").show();
        } else {
            $("#msjcp").hide();
        }
        }
    $("#cp").blur(function () {
        var longCP = $('#cp').val().length;
        if (longCP == 0) {
            $("#msjcp").hide();
        }
    });
    function validCpMx(a) {
        return a.length > 4
            && /^\d{5}$/.test(a)
            && 12345 != a
            && !/0{5}/.test(a)
            && !/1{5}/.test(a)
            && !/2{5}/.test(a)
            && !/3{5}/.test(a)
            && !/4{5}/.test(a)
            && !/5{5}/.test(a)
            && !/6{5}/.test(a)
            && !/7{5}/.test(a)
            && !/8{5}/.test(a)
            && !/9{5}/.test(a);
    }
</script>
```

Ilustración 48 Vista Datos de Pago



Representación de la vista datos de pago.

Esta vista tiene como objetivo gestionar los datos de pago pertenecientes a un proveedor, los métodos del controlador que se usaron se mostrarán en la parte de abajo. Cabe señalar que un proveedor puede tener asociadas varias formas de pago por tal motivo se implementó un Grid en donde el usuario pueda crear, editar, eliminar y visualizar las formas de pago. La funcionalidad del Grid es la siguiente, al dar clic en el botón con un signo aparece una ventana en donde se deben de proporcionar los datos necesarios a la forma de pago, una vez creado el registro se mostrará en el Grid. Ahora el usuario si lo desea podrá editar o eliminar el registro, para editarlo basta con dar clic en la fila y se mostrarán los datos o si lo desea eliminar sólo tiene que dar clic en el botón de eliminar.

Este método de tipo Json se usó para consultar y posteriormente visualizar los datos de pago asociados a un proveedor.

```
public JsonResult GetCuentas(int? idProveedor)
       {
                ViewData["BancoId"] = new SelectList(_context.Banco, "BancoId",
"Descripcion");
                ViewData["CondicionPagoId"] = new SelectList(_context.CondicionPago,
"CondicionPagoId", "Descripcion");
                ViewData["DivisaId"] = new SelectList(_context.Divisa, "DivisaId",
"Descripcion");
                ViewData["FormaPagoId"] = new SelectList(_context.FormaPago,
"FormaPagoId", "Descripcon");

                var query = (from cb in _context.CuentasBancarias
                             join b in _context.Banco
                             on cb.BancoId equals b.BancoId into bancos
                             from bank in bancos.DefaultIfEmpty()
                             join cp in _context.CondicionPago
                             on cb.CondicionPagoId equals cp.CondicionPagoId into
condicionp
                             from condicion in condicionp.DefaultIfEmpty()
                             join d in _context.Divisa
                             on cb.DivisaId equals d.DivisaId
                             join fp in _context.FormaPago
                             on cb.FormaPagoId equals fp.FormaPagoId
                             where cb.ProveedorId == idProveedor
                             select new
                             {
                                 CuentasBancariasId = cb.CuentasBancariasId,
                                 BancoId = cb.BancoId ,
                                 Banco = bank.Descripcion,
                                 ClabeInterbancaria = cb.ClabeInterbancaria,
                                 CondicionPagoId = cb.CondicionPagoId ,
                                 CondicionPago = condicion.Descripcion,
                                 DivisaId = cb.DivisaId,
                                 Divisa = d.Descripcion,
                                 FormaPagoId = cb.FormaPagoId,
                                 FormaPago = fp.Descripcion,
                                 NumeroCuenta = cb.NumeroCuenta,
                                 Sucursal = cb.Sucursal,
                                 Titular = cb.Titular,
                                 ProveedorId = cb.ProveedorId
                             }).ToList();


                return Json(query);
            }
```

Los siguientes dos métodos se emplearon para crear y editar los datos de pago.

```
[HttpPost]
        public JsonResult Create(CuentasBancarias cuentabancaria)
        {
            try
            {
                _context.Add(cuentabancaria);
                _context.SaveChanges();

                return Json(new { success = true });
            }
            catch (Exception ex)
            {
                return Json(new { success = false ,error=ex});
            }
        }
 [HttpPost]
        public JsonResult Edit(int id, CuentasBancarias cuentasbancarias)
        {
            if (id != cuentasbancarias.CuentasBancariasId)
            {
                return Json(new { success = false });
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(cuentasbancarias);
                     _context.SaveChanges();
                    return Json(new { success = true });
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!CuentasBancariasExists(cuentasbancarias.CuentasBancariasId))
                    {
                        return Json(new { success = false });
                    }
                    else
                    {
                        throw;
                    }
                }
            }
            else
            {
                return Json(new { success = false });
            }

        }
```

Y por último el método de eliminar.

```
[HttpPost]
        public JsonResult DeleteConfirmed(int id)
        {
            try
            {
                var cuentasbancarias = _context.CuentasBancarias.Single(m =>
m.CuentasBancariasId == id);
                _context.CuentasBancarias.Remove(cuentasbancarias);
                _context.SaveChanges();

                return Json(new { success = true });
            }
            catch (Exception ex)
            {
                return Json(new { success = false, error=ex });
            }
        }
```

Para este caso se usó una vista normal y otra parcial. La vista normal es a la cual se re direcciona desde el navegador, la parcial se usó para realizar todo el CRUD de los datos de pago. Para que el CRUD pudiera llevarse a cabo se tuvo que implementar un componente de JavaScript que facilitó la tarea. La forma en la que se usó código JavaScript desde la vista para mandar a llamar el componente es el siguiente.

```
<script
src="~/js/components/cuentasbancarias/controller.cuentasbancarias.js"></script>

<script type="text/javascript">
    var configCuentasCRUD = {
                urlGetInfo: '@Url.Action("GetCuentas","CuentasBancarias",new {
idProveedor = @ViewData["ProveedorId"] })',
                urlEdit: '@Url.Action("Edit", "CuentasBancarias")',
                urlCreate: '@Url.Action("Create", "CuentasBancarias")',
                urlDelete: '@Url.Action("DeleteConfirmed", "CuentasBancarias")',
                jsGridCuentasId: 'jsGridCuentas',
                SelTerenoId: 'SelProyectoId'
            };

    makeComponentGridCuentasCRUD(configCuentasCRUD);


</script>
```

En la vista parcial se creó un elemento para la visualización del Grid y también se declaró una ventana modal para dar de alta o editar los datos de pago.

```
<div id="jsGridCuentas"></div>
<div class="modal fade" id="detailsDialog" tabindex="-1" role="dialog" aria-
labelledby="gridSystemModalLabel">
<div class="modal-dialog modal-lg" role="document">
<div class="modal-content">
<div class="modal-header bg-secondary">
<h4 class="modal-title text-white" id="gridSystemModalLabel">Cuentas Bancarias</h4>
<button type="button" class="close" data-dismiss="modal" aria-label="Close"><span
aria-hidden="true">&times;</span></button>
</div>
<div class="modal-body">
<form id="detailsForm" novalidate="novalidate">
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<label for="FormaPago" class="col-form-label"> Forma de Pago</label>
@Html.DropDownList("FormaPagoId", null, "SELECCIONE", new { @class = "form-control",
id = "FormaPago", onchange = "mostrar(this.value)", required = (string)null })
</div>
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<label for="CondicionP" class="col-form-label">Condicion de Pago</label>
@Html.DropDownList("CondicionPagoId", null, "SELECCIONE", new { @class = "form-
control", id = "CondicionP" })
</div>
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<label for="banco" class="col-form-label">Banco</label>
@Html.DropDownList("BancoId", null, "SELECCIONE", new { @class = "form-control", id =
"banco" })
</div>
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<label for="sucursal" class="col-form-label">Sucursal</label>
<input name="sucursal" onKeyPress="return soloNumeros(event)" id="sucursal"
class="form-control" />
</div>
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<label for="ncuenta" class="col-form-label">Numero de Cuenta</label>
<label style="color:red">*</label>
<input name="ncuenta" onKeyPress="return soloNumeros(event)" maxlength="16"
id="ncuenta" class="form-control" />
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<label for="divisa" class="col-form-label">Divisa</label>
@Html.DropDownList("DivisaId", null, "SELECCIONE", new { @class = "form-control", id =
"divisa" })
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<input name="titular" id="titular" onkeyup="this.value=this.value.toUpperCase()"
onkeypress="return soloLetras(event)" class="form-control" required/>
</div>
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<input name="ProveedorId" type="hidden" id="pId" class="form-control "
value="@ViewData["ProveedorId"]" />                          </div>
btn btn-primary" type="submit" id="save">Guardar</button>
</div></form></div></div></div>
```

El componente JavaScript necesario para el CRUD de los datos de pago, se muestra a continuación.

```
var makeComponentGridCuentasCRUD = (function (options) {
    var config = {};
    var outside = {};
    var ElemntGrid;
    var formSubmitHandler = $.noop;
    var showDetailsDialog = function (dialogType, client) {
        $("#FormaPago").val(client.formaPagoId);
        $("#CondicionP").val(client.condicionPagoId);
        $("#banco").val(client.bancoId);
        $("#sucursal").val(client.sucursal);
        $("#ncuenta").val(client.numeroCuenta);
        $("#clabe").val(client.clabeInterbancaria);
        $("#divisa").val(client.divisaId);
        $("#titular").val(client.titular);
        $("#ProveedorId").val(client.proveedorId);
        formSubmitHandler = function () {
            saveClient(client, dialogType === "Add");
        };
        $('#detailsDialog').modal('show');
    };
    var saveClient = function (client, isNew) {
        var elemento;
        if (!isNew)
            $.extend(elemento, client);
        elemento = {
            CuentasBancariasId: null,
            FormaPagoId: $("#FormaPago").val(),
            CondicionPagoId: $("#CondicionP").val(),
            BancoId: $("#banco").val(),
            Sucursal: $("#sucursal").val(),
            NumeroCuenta: $("#ncuenta").val(),
            ClabeInterbancaria: $("#clabe").val(),
            DivisaId: $("#divisa").val(),
            Titular: $("#titular").val(),
            ProveedorId: parseInt($("#pId").val())

        };

        if (isNew)
            $.extend(client, elemento);
        else
            elemento.CuentasBancariasId = client.cuentasBancariasId;

    }
```

Se verifica el tipo de operación a realizar, ya sea de crear o editar.

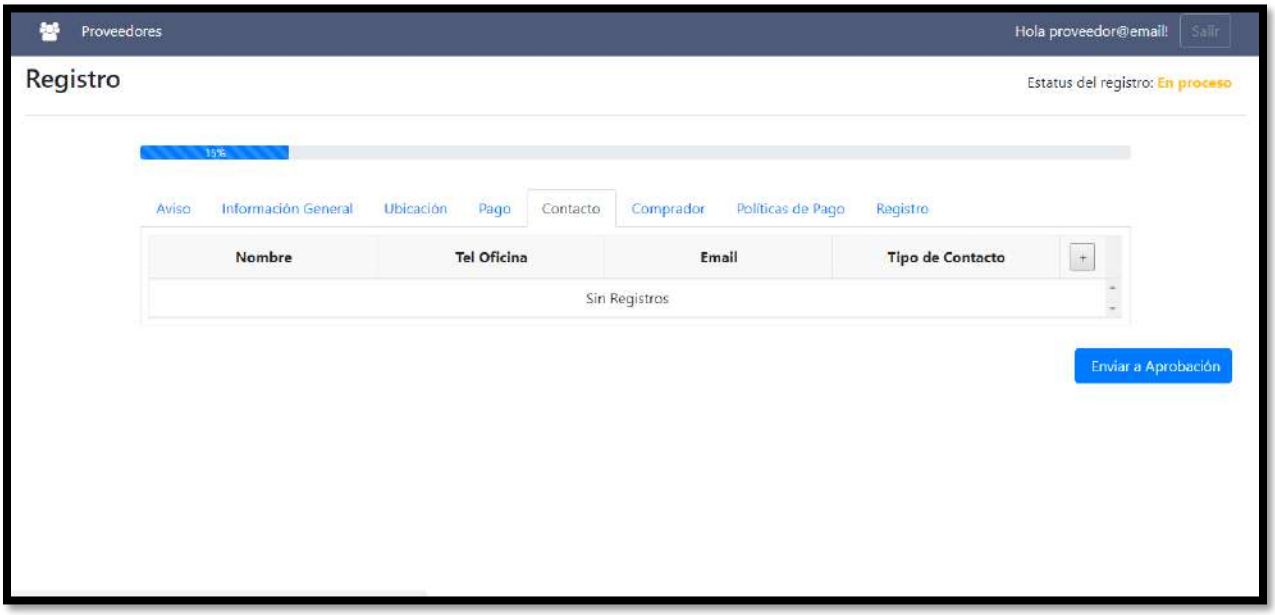
```
        if (isNew) {

            $.ajax({

                url: config.urlCreate,
                data: { cuentabancaria: elemento },
                type: 'POST',
                async: false, //blocks window close
                dataType: "json",
                success: function (data) {
                    if (data.success) {
                        $('#detailsDialog').modal('hide');
                    } else {
                        alert("Error en el Guardado");
                    }
                    jsGridGenerate();
                },
                error: function (e) {
                    $('#detailsDialog').modal('hide');
                }

            }).done(function (response) {

                $('#detailsDialog').modal('hide');
            });

        }
        else {
            $.ajax({
                url: config.urlEdit,
                data: { id: client.cuentasBancariasId, cuentasbancarias: elemento },
                type: 'POST',
                async: false, //blocks window close
                dataType: "json",
                success: function (data) {
                    if (data.success) {
                        $('#detailsDialog').modal('hide');
                    } else {
                        alert("Error al Editar");
                    }

                    jsGridGenerate();
                }
            }).done(function (response) {

            });
            $('#detailsDialog').modal('hide');
        }

    };
```

Esta función sirve para generar el Grid y realizar el CRUD.

```
var jsGridGenerate = function () {
      $("#" + config.jsGridCuentasId).jsGrid({
          height: "auto",
          width: "100%",
          sorting: true,
          paging: false,
          autoload: true,
          deleteConfirm: function (item) {
              return "¿Está seguro de que desea eliminar";
          },
          rowClick: function (args) {
              showDetailsDialog("Edit", args.item);
          },
          controller: {
              loadData: function () {
                  var d = $.Deferred();
                  $.ajax({
                      url: config.urlGetInfo,
                      dataType: "json"
                  }).done(function (response) {
                      if (response == "") {
                          $("#btnSiguienteDP").attr("disabled", "disabled");
                      } else {
                          $("#btnSiguienteDP").removeAttr("disabled", "disabled");
                      }


                      d.resolve(response);
                  });
                  return d.promise();
              },
              deleteItem: function (args) {
                  $.ajax({
                      url: config.urlDelete,
                      type: 'POST',
                      data: { id: args.cuentasBancariasId },
                      dataType: "json",
                      success: function (msg) {
                          if (msg.success == true) {
                              alert("Registro Eliminado");
                          } else {
                              alert("Error al Eliminar");
                          }
                          jsGridGenerate();
                      },
                      error: function (e) {

                      }
                  });

              },   // before controller.deleteItem
              // on done of controller.deleteItem

          },
```

```
fields: [
                {
                    name: "formaPago", type: "text",  align: "center", title: "Forma
de Pago",
                },
                {
                    name: "condicionPago", type: "text", align: "center", title:
"Condicion de Pago",
                },
                {
                    name: "banco", type: "text",  align: "center", title: "Banco",
                },
                {
                    name: "sucursal", type: "text", align: "center", title:
"Sucursal",
                },
                {
                    name: "numeroCuenta", type: "text", align: "center", title: "Núm
de Cuenta",
                }, {
                    name: "clabeInterbancaria", type: "text", align: "center", title:
"Clabe Interbancaria",
                },
                {
                    name: "titular", type: "text", align: "center", title: "Titular",
                },

                {
                    type: "control",
                    modeSwitchButton: false,
                    editButton: false,
                    headerTemplate: function () {
                        return $("<button>").attr("type", "button").text("+")
                            .on("click", function () {
                                showDetailsDialog("Add", {});
                            });
                    }
                }

            ]
        });
    };
    var formvalidate = function () {
        $("#detailsForm").validate({
            rules: {  },
            messages: { },
            submitHandler: function () {
                formSubmitHandler();
            }
        });
    };
    var init = function () {
        jsGridGenerate();
        formvalidate();
    }
    $.extend(true, config, options);init();});
```

Ilustración 49 Vista Datos de Contacto



Representación de la vista datos de contacto.

Esta vista tiene la finalidad de gestionar los contactos que el usuario asigna a un proveedor, de la misma manera que en lo datos de pago, se usó un Grid para realizar el CRUD. Los métodos utilizados son similares a los anteriores.

```
public JsonResult GetContactos(int? idProveedor)
        {
                ViewData["BancoId"] = new SelectList(_context.Banco, "BancoId",
"Descripcion");
                ViewData["CondicionPagoId"] = new SelectList(_context.CondicionPago,
"CondicionPagoId", "Descripcion");
                ViewData["DivisaId"] = new SelectList(_context.Divisa, "DivisaId",
"Descripcion");
                ViewData["FormaPagoId"] = new SelectList(_context.FormaPago,
"FormaPagoId", "Descripcion");
```

```
var query= (from c in _context.Contacto
                            join tp in _context.TipoContacto
                            on c.TipoContactoId equals tp.TipoContactoId
                            where c.ProveedorId == idProveedor
                            orderby c.ContactoId
                            select new {
                                ContactoId = c.ContactoId,
                                ApMaterno = c.ApMaterno,
                                ApPaterno = c.ApPaterno,
                                CorreoElectronico = c.CorreoElectronico,
                                Ext = c.Ext,
                                Nombre= c.Nombre,
                                NumeroFax = c.NumeroFax,
                                ProveedorAxid = c.ProveedorAxid,
                                ProveedorId = c.ProveedorId,
                                TelefonoMovil = c.TelefonoMovil,
                                TelefonoOficina = c.TelefonoOficina,
                                TelefonoParticular =  c.TelefonoParticular,
                                TipoContactoId = c.TipoContactoId,
                                Tipocontacto = tp.Descripcion
                            }).ToList();

                return Json(query);
        }
```

En las siguientes tres secciones se muestran los métodos para crear, editar y eliminar un contacto.

```
[HttpPost]
        public JsonResult Create(Contacto contacto)
        {

            try
            {
                _context.Add(contacto);
                _context.SaveChanges();

                return Json(new { success = true });
            }


            catch (Exception ex)
            {
                return Json(new { success = false,error=ex });
            }


        }
```

```
[HttpPost]
        public JsonResult Edit(int id, Contacto contacto)
        {
            if (id != contacto.ContactoId)
            {
                return Json(new { success = false });
            }

            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(contacto);
                    _context.SaveChanges();
                    return Json(new { success = true });
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!ContactoExists(contacto.ContactoId))
                    {
                        return Json(new { success = false });
                    }
                    else
                    {
                        throw;
                    }
                }
            }
            else
            {
                return Json(new { success = false });
            }
        }
```

```
[HttpPost]
        public JsonResult DeleteConfirmed(int id)
        {
            try
            {
                var contacto = _context.Contacto.Single(m => m.ContactoId == id);
                _context.Contacto.Remove(contacto);
                _context.SaveChanges();

                return Json(new { success = true });
            }
            catch (Exception ex)
            {
                return Json(new { success = false,error=ex });
            }
        }
```

En la vista se usó código JavaScript para mandar a llamar una función y pueda funcionar el CRUD.

```
var configContactosCRUD = {
       urlGetInfo: '@Url.Action("GetContactos","Contacto", new { idProveedor =
@ViewData["ProveedorId"] })',
                urlEdit: '@Url.Action("Edit", "Contacto")',
                urlCreate: '@Url.Action("Create", "Contacto")',
                urlDelete: '@Url.Action("DeleteConfirmed", "Contacto")',
                jsGridContactosId: 'jsGridContactos',
                canEdit: '@canEdit',
                SelTerenoId: 'SelProyectoId'
            };

    makeComponentGridContactosCRUD(configContactosCRUD);
```

En la vista parcial se creó una ventana modal la cual se usó para que el usuario proporcione los datos del contacto.

```
<div id="jsGridContactos"></div>
<div class="modal fade" id="detailsDialog2" tabindex="-1" role="dialog" aria-
labelledby="gridSystemModalLabel">
<div class="modal-dialog modal-lg" role="document">
<div class="modal-content">
<div class="modal-header bg-secondary text-white">
<h4 class="modal-title" id="gridSystemModalLabel">Contactos</h4>
<button type="button" class="close" data-dismiss="modal" aria-label="Close"><span
aria-hidden="true">&times;</span></button>
</div>
<div class="modal-body">
<form id="detailsForm2" novalidate="novalidate">
<div class="form-row">
<div class="form-group col-lg-6 col-md-6 col-sm-12">
<label for="TipoContacto" class="col-form-label">Tipo de Contacto <sup class="text-
danger">*</sup></label>
@Html.DropDownList("TipoContactoId", null, "seleccione", new { @class = "form-control
col-12", id = "TipoContacto", required = (string)null })
</div>
<div class="form-group col-lg-6 col-md-6 col-sm-12">
<label for="Nombre" class="col-form-label">Nombre<sup class="text-
danger">*</sup></label>
<input name="Nombre" id="Nombre" required
onkeyup="this.value=this.value.toUpperCase()" onkeypress="return soloLetras(event)"
class="form-control col-lg-12 col-md-12 col-sm-12" /></div>
```

```
</div>
<div class="form-group col-lg-6 col-md-6 col-sm-12">
<label for="app" class="col-form-label">Apellido Paterno<sup class="text-
danger">*</sup></label>
<input name="app" id="app" required onkeyup="this.value=this.value.toUpperCase()"
onkeypress="return soloLetras(event)" class="form-control col-lg-12 col-md-12 col-sm-
12" />
</div>
<div class="form-group col-lg-6 col-md-6 col-sm-12">
<label for="apm" class="col-form-label">Apellido Materno<sup class="text-
danger">*</sup></label>
<input name="apm" id="apm" required onkeyup="this.value=this.value.toUpperCase()"
onkeypress="return soloLetras(event)" class="form-control col-lg-12 col-md-12 col-sm-
12" />
</div>
<div class="form-row">
<div class="form-group col-lg-6 col-md-6 col-sm-12">
<label for="Ext" class="col-form-label">Extension</label>
<input name="Ext" id="Ext" type="tel" onKeyPress="return soloNumeros(event)"
class="form-control col-lg-12 col-md-12 col-sm-12" />
</div>
<div class="form-group col-lg-6 col-md-6 col-sm-12">
<label for="CorreoElectronico" class="col-md-2 control-label">E-mail<sup class="text-
danger">*</sup></label>
<input name="CorreoElectronico" id="CorreoElectronico" required type="email"
class="form-control col-lg-12 col-md-12 col-sm-12" />
</div>
</div>
<div class="form-row">
<div class="form-group  col-lg-6 col-md-6 col-sm-12">
<label for="TelefonoMovil" class="col-form-label">Telefono Movil</label>
<input name="TelefonoMovil" id="TelefonoMovil" maxlength="13" type="tel"
onkeyup="validarTelMovil()" onKeyPress="return soloNumeros(event)" class="form-control
col-lg-12 col-md-12 col-sm-12" />
</div>
</div>
<div class="form-group col-lg-6 col-md-6 col-sm-12">
<label for="oficina" class="col-form-label">Tel. de Oficina<sup class="text-
danger">*</sup></label>
<input name="oficina" id="oficina" maxlength="13" required type="tel"
onkeyup="validarTelOficina()" onKeyPress="return soloNumeros(event)" class="form-
control col-lg-12 col-md-12 col-sm-12" />
<div id="msjoficina" style="display:none;color:red">Introduzca un telefono
correcto</div>
</div>
</div>
<div class="form-group hidden">
<input name="idp" type="hidden" id="idp" class="form-control"
value="@ViewData["ProveedorId"]" />
</div>
<button class="btn btn-primary" type="submit" id="SaveContacto">Guardar</button>
</form>
</div>
</div>
</div>
</div>
```

La función que realiza las acciones en el Grid se muestra en la parte de abajo.

```
var makeComponentGridContactosCRUD = (function (options) {

    var config = {};
    var outside = {};
    var ElemntGrid;
    var formSubmitHandler = $.noop;

    var showDetailsDialog = function (dialogType, client) {
        $("#Nombre").val(client.nombre);
        $("#app").val(client.apPaterno);
        $("#apm").val(client.apMaterno);
        $("#oficina").val(client.telefonoOficina);
        $("#Ext").val(client.ext);
        $("#CorreoElectronico").val(client.correoElectronico);
        $("#NumeroFax").val(client.numeroFax);
        $("#TelefonoMovil").val(client.telefonoMovil);
        $("#TelefonoParticular").val(client.telefonoParticular);
        $("#TipoContacto").val(client.tipoContactoId);
        $("#ProveedorId").val(client.proveedorId);

        formSubmitHandler = function () {
            saveClient(client, dialogType === "Add");
        };

        $('#detailsDialog2').modal('show');
    };

    var saveClient = function (client, isNew) {
        var elemento;
        if (!isNew)
            $.extend(elemento, client);
        elemento = {
            ContactoId: null,
            Nombre: $("#Nombre").val(),
            ApPaterno: $("#app").val(),
            ApMaterno: $("#apm").val(),
            TelefonoOficina: $("#oficina").val(),
            Ext: $("#Ext").val(),
            CorreoElectronico: $("#CorreoElectronico").val(),
            NumeroFax: $("#NumeroFax").val(),
            TelefonoMovil: $("#TelefonoMovil").val(),
            TelefonoParticular: $("#TelefonoParticular").val(),
            TipoContactoId: $("#TipoContacto").val(),
            ProveedorId: $("#idp").val()
        };
```

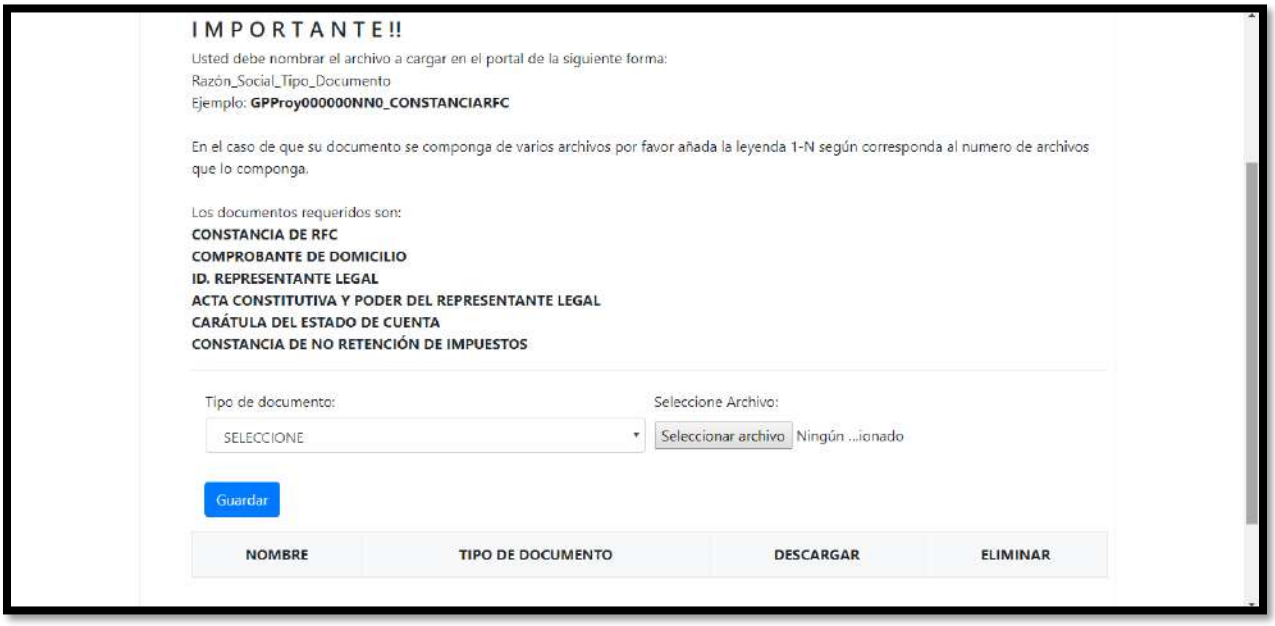
```
        if (isNew)
            $.extend(client, elemento);
        else
            elemento.ContactoId = client.contactoId;

        if (isNew) {

            $.ajax({
                url: config.urlCreate,
                data: { contacto: elemento },
                type: 'POST',
                async: false,
                dataType: "json",
                success: function (data) {
                    if (data.success) {
                        $('#detailsDialog2').modal('hide');
                    } else {
                        alert("Error al Guardar");
                    }
                    jsGridGenerate();
                },
                error: function (e) {
                    $('#detailsDialog2').modal('hide');
                }

            }).done(function (response) {
                $('#detailsDialog2').modal('hide');
            });

        }
        else {
            $.ajax({
                url: config.urlEdit,
                data: { id: client.contactoId, contacto: elemento },
                type: 'POST',
                async: false, //blocks window close
                dataType: "json",
                success: function (data) {
                    if (data.success) {
                        $('#detailsDialog2').modal('hide');
                    } else {
                        alert("Error al Editar");
                    }
                    jsGridGenerate();
                },
            }).done(function (response) {
                $('#detailsDialog2').modal('hide');
            });
            $('#detailsDialog2').modal('hide');
        }

    };
```

La siguiente función genera el Grid y se declaran algunos de sus métodos para lograr que este funcione como se necesita.

```
var jsGridGenerate = function () {
        $("#" + config.jsGridContactosId).jsGrid({
            height: "auto",
            width: "100%",
            sorting: true,
            paging: false,
            autoload: true,
            deleteConfirm: function (item) {
                return "¿Está seguro de que desea eliminar";
            },
            rowClick: function (args) {
                showDetailsDialog("Edit", args.item);
            },
            controller: {
                loadData: function () {
                    var d = $.Deferred();
                    $.ajax({
                        url: config.urlGetInfo,
                        dataType: "json"
                    }).done(function (response) {
                        if (response == "") {
                            $("#btnSiguienteC").attr("disabled", "disabled");
                        } else {
                            $("#btnSiguienteC").removeAttr("disabled", "disabled");
                        }

                        d.resolve(response);
                    });
                    return d.promise();
                },
                deleteItem: function (args) {
                    $.ajax({
                        url: config.urlDelete,
                        type: 'POST',
                        data: { id: args.contactoId },
                        dataType: "json",
                        success: function (msg) {
                            if (msg.success == true) {
                                alert("Registro Eliminado");
                            } else {
                                alert("Error al Eliminar");
                            }
                            jsGridGenerate();
                        },
                        error: function (e) {
                        }
                    });

                },
            },
```

```
fields: [
                {
                    name: "nombre", type: "text",  align: "center", title: "Nombre",
                },
                {
                    name: "apPaterno", type: "text",  align: "center", title: "Ap
Paterno",
                },
                {
                    name: "apMaterno", type: "text",  align: "center", title: "Ap
Materno",
                },
                {
                    name: "telefonoOficina", type: "text",  align: "center", title:
"Tel Oficina",
                },
                {
                    name: "correoElectronico", type: "text",  align: "center", title:
"Email",
                },
                {
                    name: "tipocontacto", type: "text",  align: "center", title: "Tipo
de Contacto",
                },
                {
                    type: "control",
                    modeSwitchButton: false,
                    editButton: false,
                    headerTemplate: function () {
                        return $("<button>").attr("type", "button").text("+")
                            .on("click", function () {
                                showDetailsDialog("Add", {});
                            });
                    }
                }

            ]
        });
    };
    var formvalidate = function () {
        $("#detailsForm2").validate({
            rules: {},
            messages: {},
            submitHandler: function () {
                formSubmitHandler();
            }
        });
    };
    var init = function () {
        jsGridGenerate();
        formvalidate();
    }

    $.extend(true, config, options);
    init();
});
```

Ilustración 50 Documentos de Inicio



Representación de la vista cargar documentos de inicio.

Este apartado tiene como objetivo proporcionar al usuario una manera de subir documentos legales que se necesitan para que un proveedor pueda ser aceptado. Los métodos usados para mostrar la vista y para realizar el proceso de carga de documentos se muestran a continuación. Primeramente, se muestra el método que sirve para renderizar la vista, recuerde que este método se usó en otras vistas en donde fue necesario adjuntar archivos.

```
public PartialViewResult AdjuntosList(int ownerId, string moduleId)
      {
            ViewData["moduleId"] = moduleId;
            ViewData["ownerId"] = ownerId;

            if (moduleId == "P")
            {
                ViewData["TipoDocumentoId"] = new
SelectList(_context.TipoDocumento.Where(x => x.IsFactura == false),
"TipoDocumentoId", "Documento");
            }
            else if (moduleId == "F")
            {
                ViewData["TipoDocumentoId"] = new
SelectList(_context.TipoDocumento.Where(x => x.IsFactura == true), "TipoDocumentoId",
"Documento");
      }
```

```
if (moduleId == "P") //Proyecto adjunto
      {
            return PartialView((from item in _context.DocProveedor
                  .Include(c => c.TipoDocumento)
                  where item.ProveedorId == ownerId
                  select new AdjuntosModelViews
                        {
                              AdjuntosId = item.AdjuntosId,
                              FileName = item.FileName,
                              ContentType = item.ContentType,
                              FileType = item.FileType,
                              TipoDocumento = item.TipoDocumento.Documento
                          }).ToList());
      }
      else if (moduleId == "F") //cluster adjunto
            return PartialView((from item in _context.DocFactura
                  .Include(c => c.TipoDocumento)
                  where item.CompraId == ownerId
                  select new AdjuntosModelViews
                        {
                              AdjuntosId = item.AdjuntosId,
                              FileName = item.FileName,
                              ContentType = item.ContentType,
                              //Content = item.Content,
                              FileType = item.FileType,
                              TipoDocumento = item.TipoDocumento.Documento
                          }).ToList());

      else
            return PartialView();
}
```

El siguiente método es el que permite guardar el documento en la base de datos, esto se logra convirtiendo el documento en un arreglo de Bytes.

```
[HttpPost]
public JsonResult UploadImageInDataBase(IFormFile files, string moduleId, int
ownerId,int TipoDocumentoId)
      {
           byte[] CoverImage = ConvertToBytes(files);

           try
           {
               if (files == null)
                   throw new ArgumentNullException(nameof(files));

               String FileExt = Path.GetExtension(files.FileName).ToUpper();
               Adjuntos Fd = new Adjuntos();
```

```
using (var reader = new StreamReader(files.OpenReadStream()))
       {
              byte[] contentAsByteArray = CoverImage;
              Fd.FileName = files.FileName;
             Fd.Content = contentAsByteArray;
             Fd.ContentType = files.ContentType;
             Fd.FileType = FileExt;
       }

       bool obj = false;
       int CompraId=0;

       switch (moduleId)
              {
                     case "P": //Proyecto proveedor
                     DocProveedor attproveedor = new DocProveedor()
                     {
                            ProveedorId = ownerId,
                            TipoDocumentoId = TipoDocumentoId,
                            FileName = Fd.FileName,
                            Content = Fd.Content,
                            ContentType = Fd.ContentType,
                            FileType = FileExt
                     };
                     _context.Add(attproveedor);
                    _context.SaveChanges();
                    obj = true;
                    break;
                     case "F": //cluster facturas
                    DocFactura attfactura = new DocFactura()
                     {
                            CompraId = ownerId,
                            TipoDocumentoId = TipoDocumentoId,
                            FileName = Fd.FileName,
                            Content = Fd.Content,
                            ContentType = Fd.ContentType,
                            FileType = FileExt
                     };
                     _context.Add(attfactura);
                     _context.SaveChanges();
                     obj = true;
                     CompraId = ownerId;
                     break;
                     default:
                            obj = false;
                     break;
                }
                return Json(new { success = obj, compraId = CompraId});
            }
            catch (Exception ex)            {

                return Json(new { success = false, error=ex });
            }
        }
```

```
private byte[] ConvertToBytes(IFormFile image)
       {
              byte[] imageBytes = null;
             var fileName = ContentDispositionHeaderValue
              .Parse(image.ContentDisposition)
              .FileName.Trim();
            Stream stream = image.OpenReadStream();
            BinaryReader reader = new BinaryReader(stream);
            imageBytes = reader.ReadBytes((int)image.Length);
            return imageBytes;
        }
[HttpGet]
public FileContentResult DownloadFile(int id)
       {
              Adjuntos fileDescription = _context.Adjuntos
              .Single(m => m.AdjuntosId == id);
              return File(fileDescription.Content, fileDescription.ContentType,
       fileDescription.FileName);
        }
        #endregion
```

El código para generar la vista es el que se visualiza en el apartado siguiente.

```
@model IEnumerable<PortalProveedoresGP.ModelViews.AdjuntosModelViews>
@{
    ViewData["Title"] = "Adjuntos List";
    var module = ViewData["moduleId"];
    var owner = ViewData["ownerId"];
}
<div class="container">
<form id="FromLoadId" class="form" enctype="multipart/form-data">
<div class="form-group">
<input type="hidden" name="moduleId" id="moduleId" class="form-control"
value="@module" />
<input type="hidden" name="ownerId" id="ownerId" class="form-control" value="@owner"
/>
</div>
<div class="form-group">
<label class="col-md-3 col-form-label">Tipo de documento:</label>
<div class="col-12 input-group">
<select name="TipoDocumentoId" id="TipoDocumentoId" class="form-control col-lg-9 col-
md-9 col-sm-12" asp-items="ViewBag.TipoDocumentoId" required>
<option value="" selected disabled >SELECCIONE</option>
</select>
</div>
</div>
<div class="form-group">
<label class="col-md-3 col-form-label">Seleccione Archivo:</label>
```

```
<div class="col-sm-12 col-lg-9 input-group">
<input type="hidden" name="moduleId" id="moduleId" class="form-control col-lg-12"
value="0" />
<input type="hidden" name="ownerId" id="ownerId" class="form-control col-lg-12"
value="1" />
<input type="file" name="files" id="files" />
</div>
</div>
<br />
<div class="form-group">
<div class="col-md-3"></div>
<div class="col-md-9 d-flex flex-row-reverse">
<div class="p-2">
<input type="button" onclick="uploadFile()" class="btn btn-primary" value="Guardar"
/>
</div>
</div>
</div>
<br />
</form>
</div>
<table class="table" id="TablaAdjuntosId">
    <thead>
        <tr>    <th>
                    Nombre
                </th>
                <th>
                    Tipo Archivo
                </th>
                <th>
                    Tipo Documento
                </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
    @foreach (var item in Model) {
            <tr>
                <td>
                    @Html.DisplayFor(modelItem => item.FileName)
                </td>
                <td>
                    @Html.DisplayFor(modelItem => item.ContentType)
                </td>
                <td>
                   <strong>@Html.DisplayFor(modelItem => item.TipoDocumento)</strong>
                </td>
                <td>
                    <a asp-action="DownloadFile" asp-route-
id="@item.AdjuntosId">Descargar</a>
                </td>
            </tr>
    }
    </tbody>
</table>
```

En la vista se usó código JavaScript como mecanismo para enviar los datos de la vista al controlador.

```
<script type="text/javascript">
    function uploadFile() {
        if ($("#TipoDocumentoId").val() != null) {
            var formData = new FormData(document.getElementById("FromLoadId"));
            var fileSelect = document.getElementById('files');
            var files = fileSelect.files;
            if (files.length != 0) {
                formData.append("files", files[0], files[0].name);
                formData.append("moduleId", $("#moduleId").val());
                formData.append("ownerId", $("#ownerId").val());
                $.ajax({
                    url: '@Url.Action("UploadImageInDataBase","ManageFile")',
                    type: "post",
                    dataType: "json",
                    data: formData,
                    cache: false,
                    contentType: false,
                    processData: false,
                    success: function (data) {
                        if (data.success) {
                            var compraid = data.compraId;
                            if (compraid > 0) {
                                docInfo.refrescarInfo(compraid);
                            } else {
                                docInfo.refrescarInfo();
                            }
                        } else {
                            alert("No se pudo Guardar");
                        }
                    },
                    error: function (err) {
                    }
                });
            } else {
                alert("Debe Elegir un Archivo");
            }

        } else {
            alert("Eliga Un Tipo de Documento");

        }
       }
</script>
```

Ilustración 51 Vista Crear Orden de Compra

Representación de la vista crear orden de compra.

Esta vista tiene como objetivo permitir al usuario proveedor dar de alta una orden de compra. El modelo de la orden de compra es el que aparece en la siguiente sección.

```
public partial class Compra
    {
        public Compra()
        {
            DocumentosFactura = new HashSet<DocumentosFactura>();
            PartidasCompra = new HashSet<PartidasCompra>();
        }
        public int CompraId { get; set; }
        public string NoCompra { get; set; }
        public int? ProveedorId { get; set; }
        public int? TipoCompraId { get; set; }
        public string Uuid { get; set; }
        public string Folio { get; set; }
        public string Serie { get; set; }
        public int? CompradorId { get; set; }
        public DateTime FechaInicio { get; set; }
        public string AreaAxid { get; set; }
        public string ContratoAxid { get; set; }
        public string ProvedorAxid { get; set; }
        public string OrdenAxid { get; set; }
        public string RespuestaEnvioAx { get; set; }
        public DateTime FechaEnvioAx { get; set; }
        public virtual ICollection<DocumentosFactura> DocumentosFactura { get; set; }
        public virtual ICollection<PartidasCompra> PartidasCompra { get; set; }
        public virtual Comprador Comprador { get; set; }
        public virtual Proveedores Proveedor { get; set; }
        public virtual TipoCompra TipoCompra { get; set; }
    }
}
```

Los métodos usados en el controlador para mostrar la vista y guardar los datos que el usuario proporcione son los siguientes.

```
[Authorize (Policy = "CompraCrear")]
public IActionResult Create(int? id)
        {
            ViewData["Compradores"] = new SelectList(_context.ProveedorComprador
                .Include(c => c.Comprador)
                .Where(c => c.ProveedorId == id)
                .Select(c => new
                {
                    CompradorId = c.CompradorId,
                    Nombre = c.Comprador.Nombre
                }),
                "CompradorId",
                "Nombre",
                null);
            ViewData["ProveedorId"] = new SelectList(_context.Proveedores,
"ProveedorId", "RazonSocial");
            ViewData["TipoCompraId"] = new SelectList(_context.TipoCompra,
"TipoCompraId", "Descripcion");
            ViewData["ProveedorId"] = id;

            var proveedor = _context.Proveedores
                 .Single(p => p.ProveedorId == id);
            if (proveedor == null)
            {
                return NotFound();
            }
            ViewData["RazonSocial"] = proveedor.RazonSocial;

            return PartialView();
        }
```

El método anterior sirve para renderizar la vista y pasarle, note que el controlador pasa objetos a la vista, con la finalidad de que esta pueda funcionar adecuadamente. En la sección de abajo se muestra el método que permite guardar los datos.

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult>
Create([Bind("CompraId,NoCompra,ProveedorId,TipoCompraId,Uuid,Folio,Serie,CompradorId,
FechaInicio,AreaAxid,ContratoAxid,ProvedorAxid,OrdenAxid,RespuestaEnvioAx,FechaEnvioAx
")] Compra compra)
        {
            if (ModelState.IsValid)
            {   _context.Add(compra);
                await _context.SaveChangesAsync();
                return RedirectToAction("Index", new { proveedorid =
compra.ProveedorId });
            }
            ViewData["CompradorId"] = new SelectList(_context.Comprador,
"CompradorId", "CompradorId", compra.CompradorId);
            ViewData["ProveedorId"] = new SelectList(_context.Proveedores,
"ProveedorId", "ProveedorId", compra.ProveedorId);
            ViewData["TipoCompraId"] = new SelectList(_context.TipoCompra,
"TipoCompraId", "TipoCompraId", compra.TipoCompraId);
            return PartialView(compra);
        }
```

Los métodos que a continuación se muestran se utilizan para editar una orden de compra.

```
[Authorize (Policy = "CompraEditar")]

public async Task<IActionResult> Edit(int? id)
        {
            if (id == null)
            {
                return NotFound();
            }
            ViewData["CompraId"] = id;
            var compra = await _context.Compra.SingleOrDefaultAsync(m => m.CompraId ==
id);
            if (compra == null)
            {
                return NotFound();
            }
            ViewData["CompradorId"] = new SelectList(_context.Comprador,
"CompradorId", "Nombre", compra.CompradorId);
            ViewData["ProveedorId"] = new SelectList(_context.Proveedores,
"ProveedorId", "RazonSocial", compra.ProveedorId);
            ViewData["TipoCompraId"] = new SelectList(_context.TipoCompra,
"TipoCompraId", "Descripcion", compra.TipoCompraId);
            return PartialView(compra);
}
```

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id,
[Bind("CompraId,NoCompra,ProveedorId,TipoCompraId,Uuid,Folio,Serie,CompradorId,FechaIn
icio,AreaAxid,ContratoAxid,ProvedorAxid,OrdenAxid,RespuestaEnvioAx,FechaEnvioAx")]
Compra compra)
        {
            if (ModelState.IsValid)
            {
                try
                {
                    _context.Update(compra);
                    await _context.SaveChangesAsync();
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!CompraExists(compra.CompraId))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
                return RedirectToAction("Index");
            }
            ViewData["CompradorId"] = new SelectList(_context.Comprador,
"CompradorId", "CompradorId", compra.CompradorId);
            ViewData["ProveedorId"] = new SelectList(_context.Proveedores,
"ProveedorId", "ProveedorId", compra.ProveedorId);
            ViewData["TipoCompraId"] = new SelectList(_context.TipoCompra,
"TipoCompraId", "TipoCompraId", compra.TipoCompraId);
            return PartialView(compra);
        }
```

Para que el usuario pueda visualizar los datos de la orden de compra, es necesario usar un método que permite consultar los datos desde la base de datos y posteriormente pasarlos a la vista.

```
 [Authorize (Policy = "CompraVer Detalle")]
public async Task<IActionResult> Details(int? id, int? proveedorid)
      {
            ViewData["CompraId"] = id;
            ViewData["Compradores"] = new SelectList(_context.ProveedorComprador
                .Include(c => c.Comprador).Where(c => c.ProveedorId == proveedorid)
                .Select(c => new{CompradorId = c.CompradorId,
                 Nombre =   c.Comprador.Nombre
                }),
                "CompradorId","Nombre",null);
            ViewData["StatusOC"] = "";
            try
            {
                var statusOC = _context.DocFactura.Include(f => f.Estatus)
                .Single(f => f.CompraId == id && f.Estatus.Descripcion ==
"Aprobado");
                ViewData["StatusOC"] = "Si";
            }
            catch {
                ViewData["StatusOC"] = "";

            }

            if (id == null)
            {
                return NotFound();
            }

            var compra = await _context.Compra
                .Include(c => c.Comprador)
                .Include(c => c.Proveedor)
                .Include(c => c.TipoCompra)
                .SingleOrDefaultAsync(m => m.CompraId == id);
            if (compra == null)
            {
                return NotFound();
            }

            return PartialView(compra);
        }
```

Una vez mostrados los métodos implementados para la gestión de las órdenes de compra, se dará a conocer el código usado en las vistas de crear, editar y ver detalle, respectivamente.

Vista Crear Orden de Compra.

```
@model PortalProveedores.Models.Compra
<form asp-action="Create">
<div class="form-horizontal">
<h4>Nueva Compra</h4>
<div asp-validation-summary="ModelOnly" class="text-danger"></div>
<div class="form-row col-lg-12 ">
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<label asp-for="ProveedorId" class="col-form-label">Proveedor</label>
<div class="col-md-10">
<input type="hidden" asp-for="ProveedorId" class="form-control"
value="@ViewData["ProveedorId"]" />
<label class="form-control-label"><b>@ViewData["RazonSocial"]</b></label>
</div>
</div>
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<label asp-for="NoCompra" class="col-form-label"># Orden de Compra</label>
<div class="col-md-10">
<input asp-for="NoCompra" id="Ncompra" class="form-control" required/>
</div>
</div>
</div>
<div class="form-row col-lg-12 ">
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<label asp-for="TipoCompraId" class="col-form-label">Tipo Compra</label>
<div class="col-md-10">
<select asp-for="TipoCompraId" class="form-control" required asp-
items="ViewBag.TipoCompraId">
<option value="" selected disabled>SELECCIONE</option>
</select>
</div>
</div>
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<label asp-for="FechaInicio" class="col-form-label">Fecha</label>
<div class="col-md-10">
<input asp-for="FechaInicio" type="date" class="form-control" required />
</div>
</div>
</div>
<div class="form-row col-lg-12 ">
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<label asp-for="CompradorId" class="col-form-label">Comprador</label>
<div class="col-md-10">
<select asp-for="CompradorId" class="form-control" asp-items="ViewBag.Compradores"
required id="compradorid">
</div>
<div class="form-group">
<div class="col-md-offset-2 col-md-10">
<input type="submit" value="Guardar" class="btn btn-primary" />
<input type="button" id="btnCancel" value="Cancelar" class="btn btn-secondary">
</div>
</div>
</form>
```

Vista Editar Orden de Compra.

```
@model PortalProveedores.Models.Compra
<div class="col-md-12 paddingSuperior2">
<form asp-action="Edit">
<div class="form-horizontal">
<div asp-validation-summary="ModelOnly" class="text-danger"></div>
<input type="hidden" id="CompraId" asp-for="CompraId" />
<div class="form-row col-lg-12 ">
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<label asp-for="TipoCompraId" class="col-form-label">Tipo de Compra</label>
<div class="col-md-10">
<select asp-for="TipoCompraId" id="TipoCompraId" class="form-control" required asp-
items="ViewBag.TipoCompraId">
<option selected disabled>SELECCIONE</option>
</select>
</div></div>
<div class="form-row col-lg-12 ">
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<label asp-for="Folio" class="col-form-label"></label>
<div class="col-md-10">
<input asp-for="Folio" class="form-control" id="folio"
onkeyup="this.value=this.value.toUpperCase()"  required />
</div></div>
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<label asp-for="Serie" class="col-form-label"></label>
<div class="col-md-10">
<input asp-for="Serie" class="form-control" id="serie"
onkeyup="this.value=this.value.toUpperCase()"  />
</div></div></div>
<div class="form-row col-lg-12 ">
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<label asp-for="FechaInicio" class="col-form-label">Fecha</label>
<div class="col-md-10">
<input asp-for="FechaInicio" type="date" class="form-control" id="fecha" required/>
</div></div>
<div class="form-group col-lg-6 col-md-6 col-sm-6">
<label asp-for="CompradorId" class="col-form-label">Comprador</label>
<div class="col-md-10">
<select asp-for="CompradorId" class="form-control" id="CompradorId" required asp-
items="ViewBag.CompradorId">
<option selected disabled>SELECCIONE</option>
</select>
</div></div></div>
<input type="submit" value="Guardar" class="btn btn-primary" />
@*si no hay xml valido desactivar *@
<button id="EnviaraSuministros"  class="btn btn-dark">Enviar a validar</button>
<input type="button" id="btnCancel" value="Cancelar" class="btn btn-secondary">
<div id="msjId" class="alert alert-info" role="alert">
</div></div>
</form>
</div>
```

Vista ver Detalle de Orden de Compra.

```
@model PortalProveedores.Models.Compra
<div class=" col-12 bg-light text-center text-secondary border mt-3 pt-2 pb-1">
    <h4>DETALLE</h4>
</div>
<input id="SelCompraId2" type="hidden"   value="@Model.CompraId"/>

<div class="col-md-12 paddingSuperior2">
    <div class="form-row col-lg-12 ">
        <div class="form-group col-lg-6 col-md-6 col-sm-6">
            <label class="col-form-label">Proveedor: </label>
            <label class="col-form-label"><b>@Html.DisplayFor(model =>
model.Proveedor.RazonSocial)</b></label>
        </div>
        <div class="form-group col-lg-6 col-md-6 col-sm-6">
            <label class="col-form-label"># OC / ESTIMACI&Oacute;N: </label>
            <label class="col-form-label"><b>@Html.DisplayFor(model =>
model.NoCompra)</b></label>
        </div>
    </div>
    <div class="form-row col-lg-12 ">
        <div class="form-group col-lg-6 col-md-6 col-sm-6">
            <label class="col-form-label">Tipo de Compra: </label>
            <label class="col-form-label"><b>@Html.DisplayFor(model =>
model.TipoCompra.Descripcion)</b></label>
        </div>
        <div class="form-group col-lg-6 col-md-6 col-sm-6">
            <label class="col-form-label">UUID: </label>
            <label class="col-form-label"><b>@Html.DisplayFor(model =>
model.Uuid)</b></label>
        </div>
    </div>
    <div class="form-row col-lg-12 ">
        <div class="form-group col-lg-6 col-md-6 col-sm-6">
            <label class="col-form-label">Folio: </label>
            <label class="col-form-label"><b>@Html.DisplayFor(model =>
model.Folio)</b></label>
        </div>
        <div class="form-group col-lg-6 col-md-6 col-sm-6">
            <label class="col-form-label">Serie: </label>
            <label class="col-form-label"><b>@Html.DisplayFor(model =>
model.Serie)</b></label>
        </div>
    </div>
    <div class="form-row col-lg-12 ">
        <div class="form-group col-lg-6 col-md-6 col-sm-6">
            <label class="col-form-label">Fecha: </label>
            <label class="col-form-label"><b>@Html.DisplayFor(model =>
model.FechaInicio)</b></label>
        </div>
```

```
@if (User.IsInRole("Proveedor"))
{
    if (ViewData["StatusOC"].ToString() == "Si")
    {
            <div class="form-group col-lg-6 col-md-6 col-sm-6">
                <label class="col-form-label">Comprador: </label>
                <label class="col-form-label"><b>@Html.DisplayFor(model =>
model.Comprador.Nombre)</b></label>
            </div>
    }
    else
    {
            <div class="form-group col-lg-6 col-md-6 col-sm-6">
                <label class="col-form-label">Comprador: </label>
                <div class="col-md-10">
                    <select asp-for="CompradorId" class="form-control" asp-
items="ViewBag.Compradores" id="compradorid">
                        <option value="" disabled selected>SELECCIONE</option>
                    </select>
                </div>
            </div>
    }
}
else
{
    <div class="form-group col-lg-6 col-md-6 col-sm-6">
        <label class="col-form-label">Comprador: </label>
        <label class="col-form-label"><b>@Html.DisplayFor(model =>
model.Comprador.Nombre)</b></label>
    </div>
}
    </div></div>
<div class="col-md-offset-2 col-12 p-2 bg-light border border-top-0 border-lilght">

    @*si no hay xml valido desactivar *@
    <button id="EnviaraSuministros" class="btn btn-primary mt-2 mb-2">Enviar a
validar</button>
    <div id="msjId" class="alert alert-primary col-12" role="alert">
    </div>
</div>
<div class="container text-center border border-top-0 border-lilght col-12 pt-2 pb-2
clearfix">
    <div class="float-left col-lg-6 col-md-6 col-sm-12">
        <button id="verpartidas" class="btn btn-outline-info">+</button>
        <label class="form-control-label" for="verpartidas">Detalle Partidas</label>
    </div>
    <div class="float-left col-lg-6 col-md-6 col-sm-12">
        <button id="verfacturas" class="btn btn-outline-info">+</button>
        <label class="form-control-label" for="verfacturas">Facturas Cargadas</label>
    </div>
</div>
</div>
```

En esta vista también se implementó código JavaScript, parte de este código es el que a continuación se muestra.

```
<script>
    var configFiles = {
        adjuntosListURL: '@Url.Action("TableFacturas", "ManageFacturas")',
                    divArchivosId: 'divFacturas2',
                    SelCompraId: 'SelCompraId2',
                    moduleId:"F"
        };
    var docInfo = makeComponentCRUDFiles(configFiles);
    var PatidasCompra = {
        GetInfo: '@Url.Action("GetPartidas", "PartidasCompra")',
        CompraId: 'SelCompraId2',
        };
    MakeGridPartidas(PatidasCompra);
    $("#verfacturas").on("click", function () {
        $("#FacturasM").modal("show");
    });
    $("#verpartidas").on("click", function () {
        $("#MPartidasC").modal("show");
    });
    var url =  '@Url.Action("Post", "Factura")';
    $.ajax({
        url: url,
        data: { estimacionID: $("#SelCompraId2").val() },
        type: 'POST',
        async: false, //blocks window close
        dataType: "json",
        success: function (data) {

            if (data.success) {
                $("#EnviaraSuministros").show();
                $('#msjId').empty();
                $('<strong>' + data.msj.value + '</strong>').appendTo('#msjId');
            }
            else {
                $("#EnviaraSuministros").hide();
                $('#msjId').empty();
                $('<strong>' + data.msj + '</strong>').appendTo('#msjId');
            }
        },
        error: function (e) {

            $("#EnviaraSuministros").hide();
            $('#msjId').empty();
            $('<strong>' + data.msj + '</strong>').appendTo('#msjId');
        }

    }).done(function (response) {

        });

</script>
```

5.4 Prueba de Aceptación del Sistema

Las pruebas de aceptación son pruebas de caja negra definidas por el cliente para cada historia de usuario, y tienen como objetivo asegurar que las funcionalidades del sistema cumplen con lo que se espera de ellas. En efecto, las pruebas de aceptación marcan el camino a seguir en cada iteración, indicándole al equipo de desarrollo hacia dónde tiene que ir y en qué puntos o funcionalidades debe poner el mayor esfuerzo y atención.

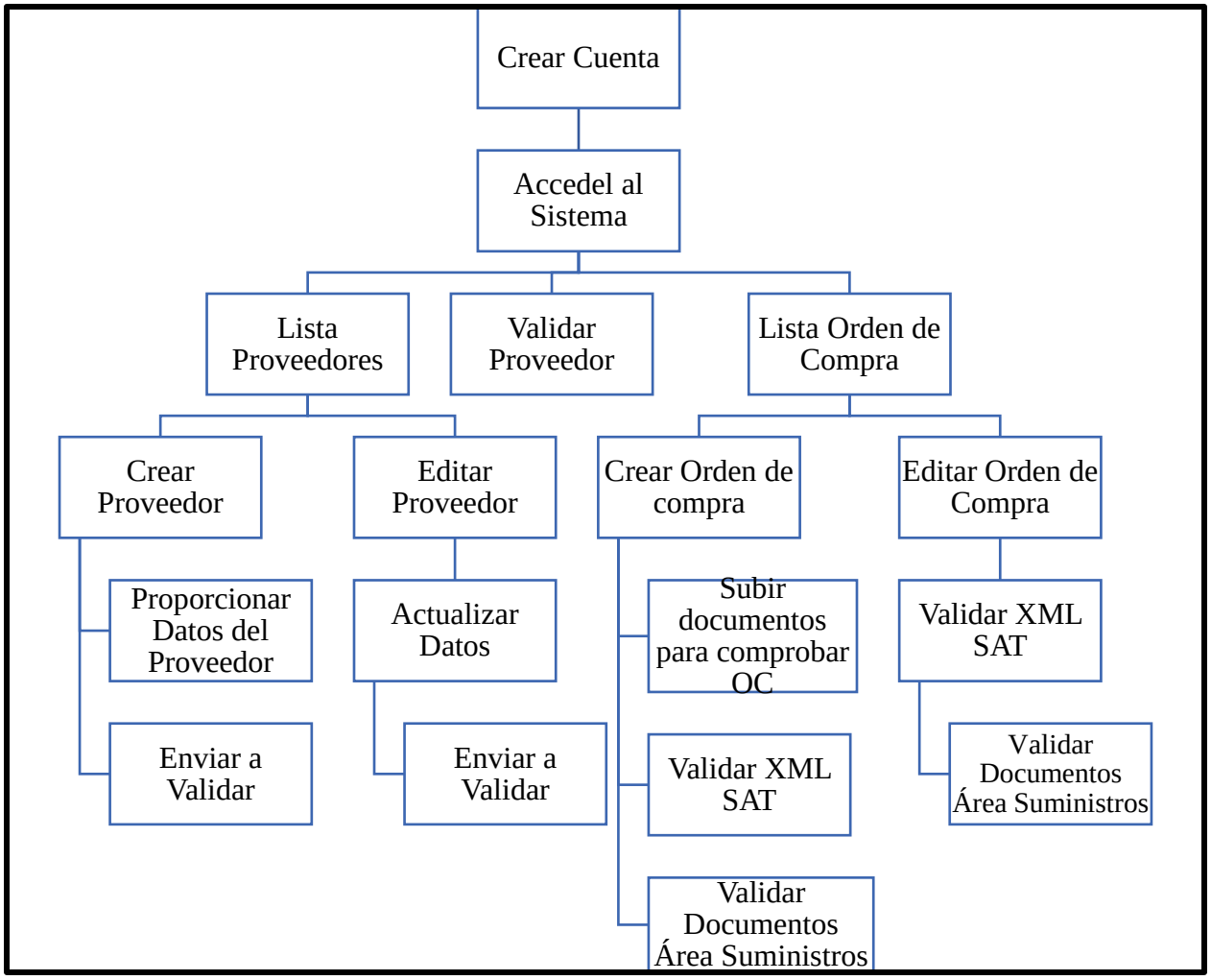
“Las pruebas de aceptación permiten al cliente saber cuándo el sistema funciona, y que los programadores conozcan qué es lo que resta por hacer.”, según Jeffries, (2000).

El objetivo de este tipo de prueba es comprobar si el cliente está satisfecho con el sistema desarrollado y si el sistema cumple con sus expectativas y necesidades. Para ello se puso a prueba con los usuarios finales de la empresa inmobiliaria, ingresando tipos de datos que ellos utilizan, verificando los permisos que deben de tener los distintos roles del sistema, la validación de facturas ante el SAT y las notificaciones vía correo electrónico. Finalmente, los resultados fueron analizados y aprobados por los directivos de la empresa inmobiliaria, recalcando la utilidad, comodidad y facilidad de usar el sistema desarrollado, siendo una excelente herramienta tanto para los empleados de la empresa, así como también para sus clientes, debido a que permite la comunicación con otros sistemas para poder llevar a cabo el flujo de trabajo que se demanda.

VI. Resultados

6.1 Mapa de Navegación

Ilustración 52 Mapa de Navegación



Representación del Mapa de Navegación del Sistema.

1. Crear Cuenta
    - Llenar campos: correo, contraseña, confirmar contraseña.
    - Botón registrar: clic en el botón para crear cuenta de acceso
2. Acceder al sistema
    - Llenar campos: correo y contraseña.
    - Botón Log in: clic en el botón para ingresar al sistema.
3. Lista de Proveedores
    a) Crear proveedor
        - Llenar campos: tipo de persona, tipo de empresa, razón, social, etc.
        - Botón guardar: clic en el botón para crear el registro.
        - Proporcionar datos del proveedor:
        - Llenar campos: aceptar aviso, datos de generales, datos de ubicación, datos de pago, datos de contacto, aceptar políticas, subir documentos de inicio.
        - Botón Enviar a aprobación: clic en el botón para enviar a validar toda la información del proveedor.
    a) Editar Proveedor
        - Actualizar campos: aceptar aviso, datos de generales, datos de ubicación, datos de pago, datos de contacto, aceptar políticas, subir documentos de inicio.
        - Botón Enviar a aprobación: clic en el botón para enviar a validar toda la información del proveedor.

4. Validar proveedor
    - Botón Aprobar: clic en el botón para aprobar el registro del proveedor.
    - Botón rechazar: clic en el botón para rechazar el registro del proveedor.

5. Lista Orden de Compra
    a) Crear Orden de Compra
        - Llenar campos: orden de compra, tipo de compra, fecha, comprador y subir documento orden de compra.
        - Botón Guardar: clic en el botón para guardar el registro.
        - Botón Cancelar: clic en el botón para cancelar el registro.

        I. Subir documentos
            - Adjuntar archivos: XML, orden de compra, factura pdf.
            - Botón Guardar: clic en el botón para subir los documentos.

        II. Validar XML SAT
            - Cuando el usuario sube el documento XML este se envía a validar en el SAT.

- Si el XML es correcto se habilita el botón enviar a validar al área de suministros.
- Botón Enviar a Validar: clic en el botón para enviar a validar.

III. Validar Documentos Área Suministros

- Botón Aprobar: clic en el botón para aprobar los documentos adjuntos que el proveedor subió.
- Botón Rechazar: clic en el botón para rechazar los documentos adjuntos que el proveedor subió.

a) Editar Orden de Compra:

- Llenar campos: orden de compra, tipo de compra, fecha, comprador.
- Botón Guardar: clic en el botón para guardar los cambios.
- Botón Cancelar: clic en el botón para cancelar el registro.

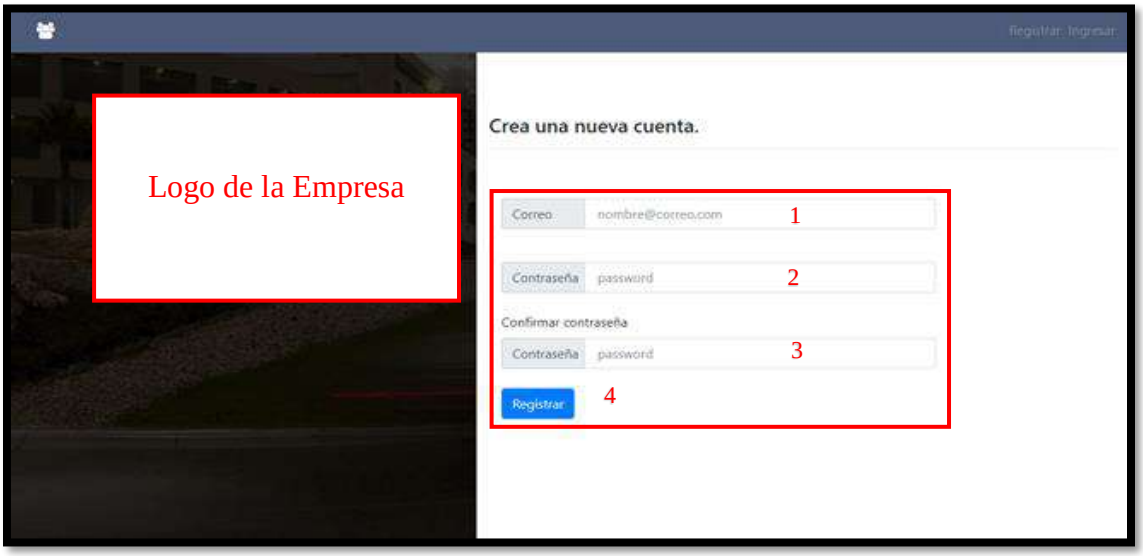
I. Validar XML SAT

- Cuando el usuario sube el documento XML este se envía a validar en el SAT.
- Si el XML es correcto se habilita el botón enviar a validar al área de suministros.
- Botón Enviar a Validar: clic en el botón para enviar a validar.

II. Validar Documentos Área Suministros

- Botón Aprobar: clic en el botón para aprobar los documentos adjuntos que el proveedor subió.
- Botón Rechazar: clic en el botón para rechazar los documentos adjuntos que el proveedor subió.

Ilustración 53 Creación de Cuenta de Acceso



Representación creación de Cuenta de Acceso.

Ilustración 54 Ingresar al Sistema

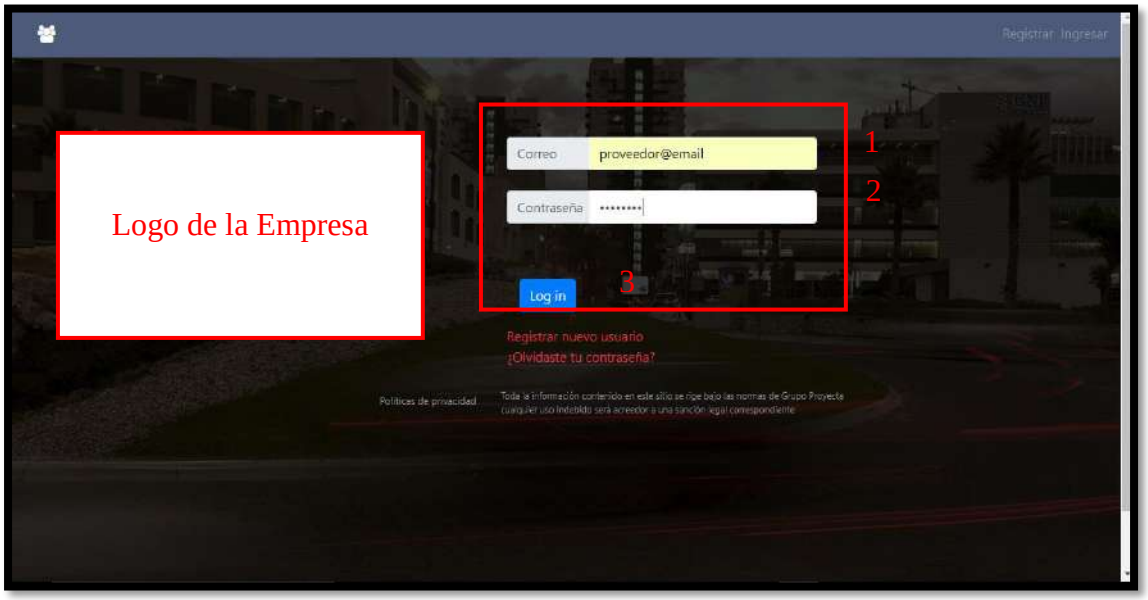
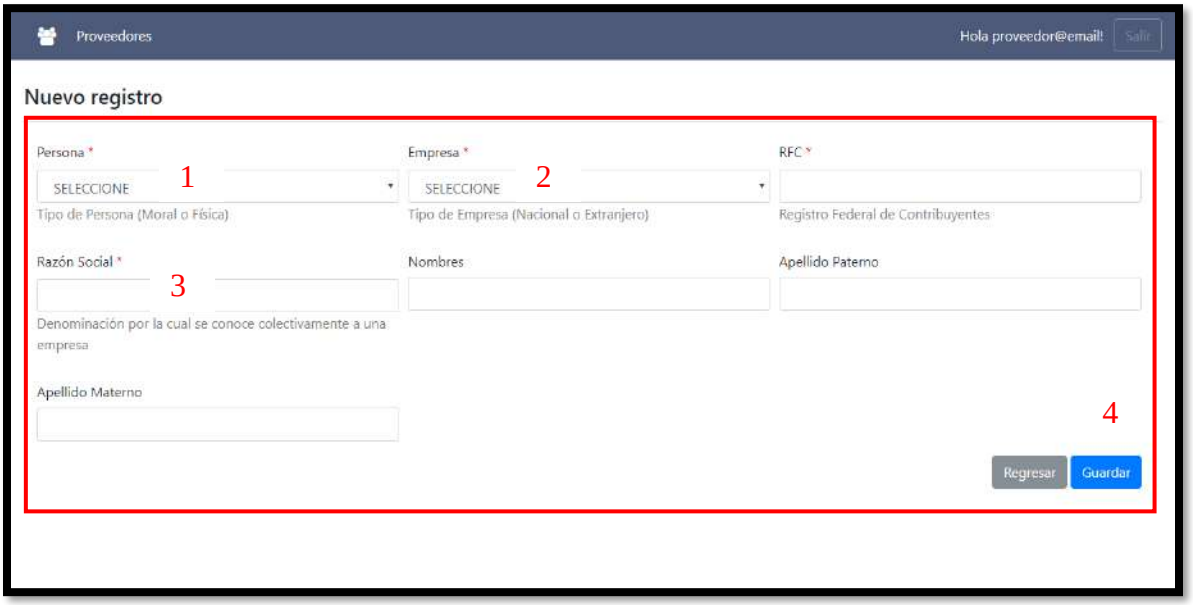


Ilustración 55 Tabla de Proveedores

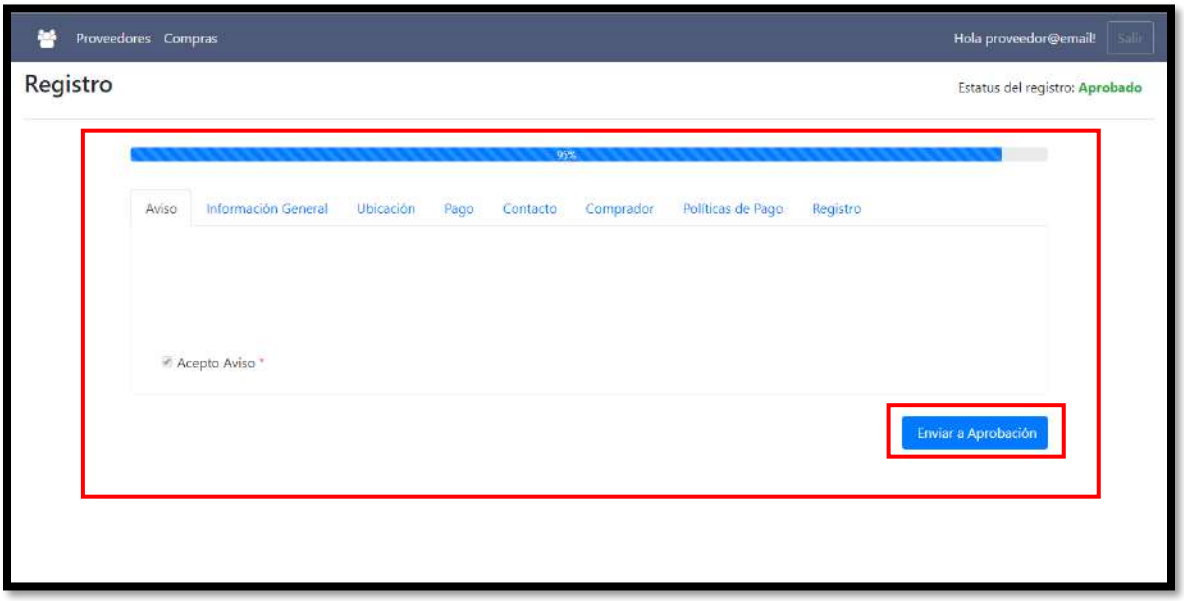
Representación tabla de Proveedores.

Ilustración 56 Crear Proveedor



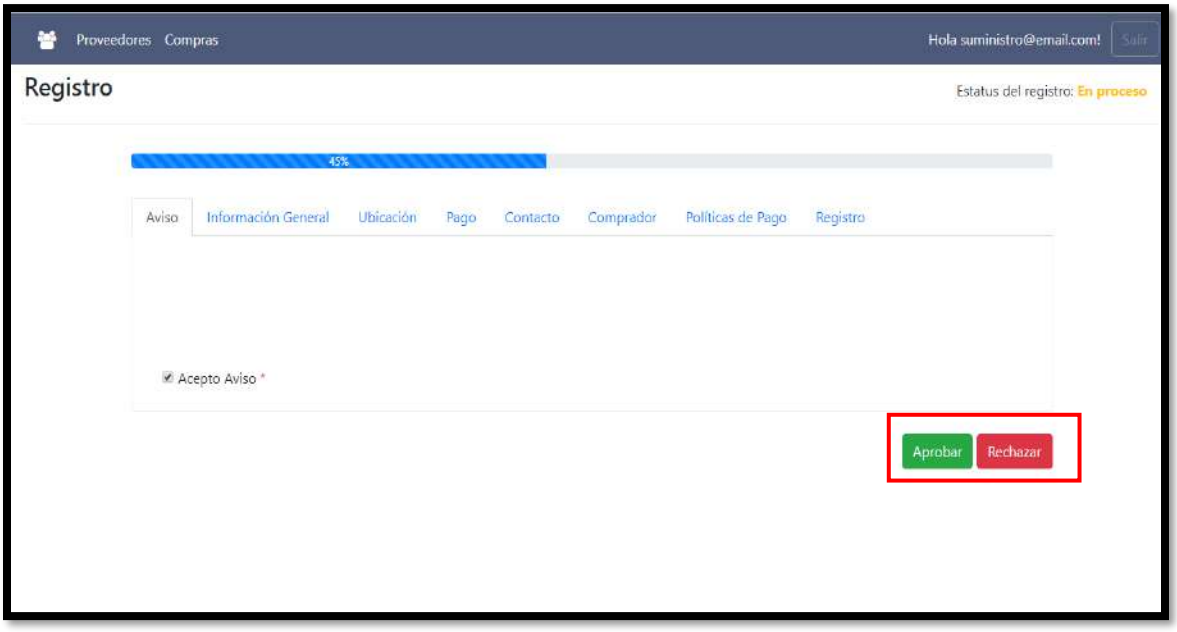
Representación crear Proveedor.

Ilustración 57 Alta de Datos del Proveedor



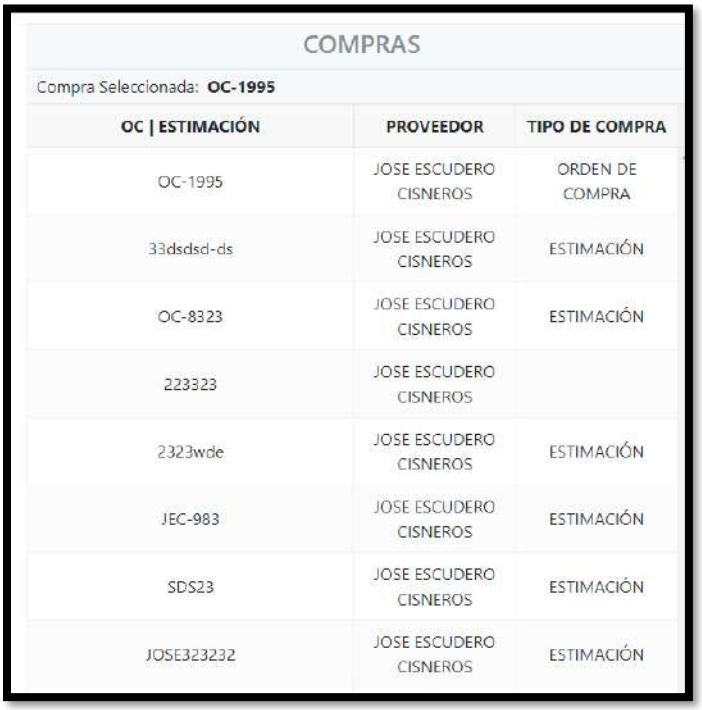
Representación ingresar datos del Proveedor.

Ilustración 58 Validar Proveedor



Representación aprobar o rechazar Proveedor.

Ilustración 59 Tabla de Órdenes de Compras



Representación tabla de Órdenes de Compra.

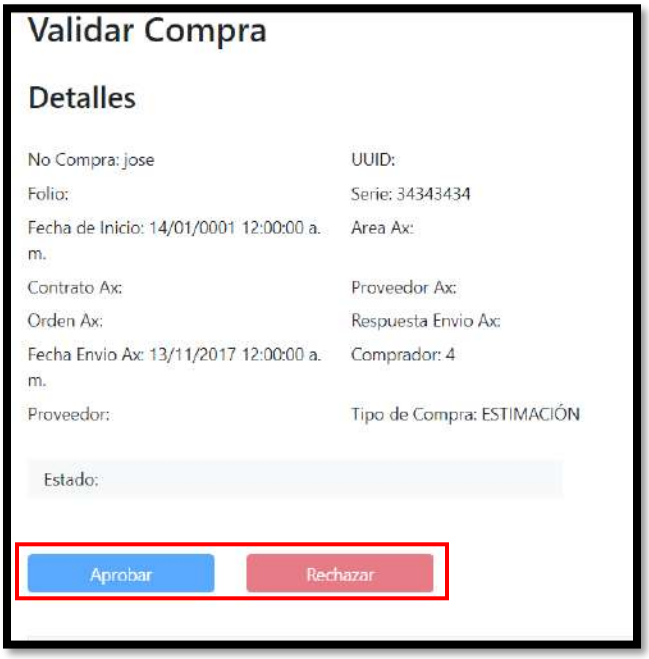
Ilustración 60 Crear Orden de Compra

Representación Crear Orden de Compra.

Ilustración 61 Subir Facturas

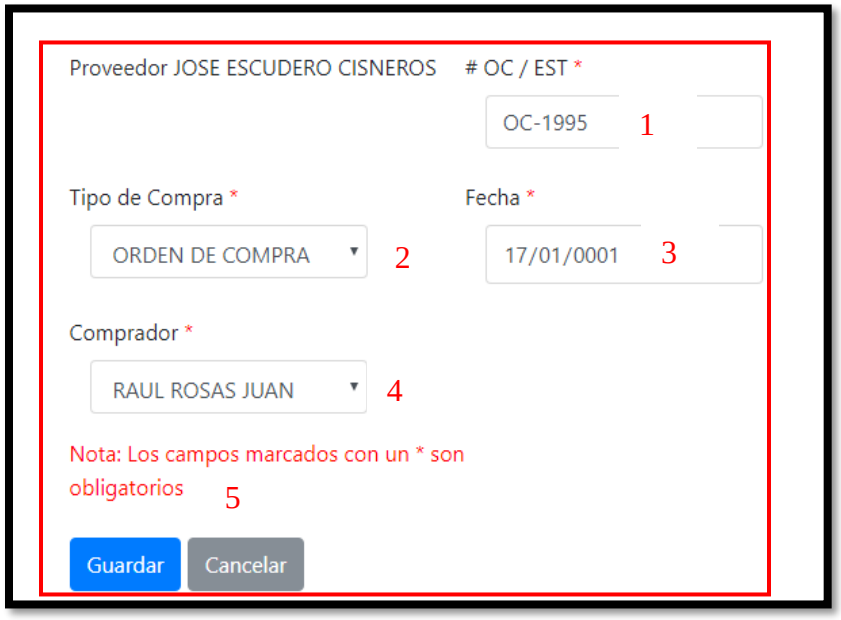
Representación Adjutar Facturas.

Ilustración 62 Validar Orden de Compra



Representación Validar Orden de Compra.

Ilustración 63 Editar Orden de Compra



Representación Editar Orden de Compra.

VII. Conclusiones

La utilización de la Metodología XP (Programación Extrema) fue una guía importante a seguir en cada una de las tareas realizadas, debido a que proporciona de manera ordenada cada una de sus fases y actividades en el desarrollo de un proyecto de software.

La arquitectura MVC ofrece una forma de programación dividida en tres principales componentes, logrando una mejor forma de estructuración del código y por lo tanto mayor comprensión para los desarrolladores. Respecto a las herramientas de desarrollo utilizadas, la utilidad de estas fue de suma importancia para cumplir con cada uno de los requisitos del sistema. Con respecto a la base de datos, el uso de Entity Framework, junto con la plataforma ASP.NET MVC, otorgan una mayor velocidad de desarrollo, porque la conexión a esta no necesita de componentes adicionales para acceder a los datos, por lo tanto, esto mejora la velocidad de respuesta.

Respecto a la solución entregada, al analizar los objetivos planteados de la toma de requerimientos, el Sistema logra cumplir las metas establecidas. Esto es, que el sistema permita almacenar diferentes tipos de datos y comunicarse con otros sistemas para llevar a cabo los procedimientos que se pidieron en la toma de requerimientos. El sistema permite la comunicación con otro sistema para que este pueda insertar en la base de datos, así como también se le permite al usuario adjuntar archivos tales como XML los cuales son validados ante el SAT, esto logra que el sistema sea apto y cumpla con las necesidades del cliente.

VIII. Bibliografía

Álvarez, M. (2014). *Qué es MVC.* Recuperado de https://desarrolloweb.com/articulos/que-es-mvc.html

Cárdenas, C. (2015). *Sistema de Comunicaciones Internas de la Escuela de Ingeniería en Computación (CI-IC).* Escuela de Ingeniería en Computación. Universidad Austral de Chile.

Centeno, J. (2014). *Sistema de Gestión del proceso de Valuación de Inmuebles (Caso: Integra Ingeniería Inmobiliaria S.A. de C.V.).* Unidad Profesional Interdisciplinaria de Ingeniería y Ciencias Sociales y Administrativas. Instituto Politécnico Nacional.

Cepeda, M. (2014). *Implementación de un Sistema Informático para la Gestión Académica de la Unidad Educativa Intercultural Bilingüe "Corazón de la Patria".* Facultad de Sistemas Mercantiles. Universidad Regional Autónoma de los Andes.

Choque, J. y Nazar, F. (2015). *Diseño e Implementación del Sistema de Gestión Web que integra componentes de los Sílabos de cursos aplicando Laravel en el Proceso de Control de Avance Silábico de la Universidad Privada de Tacna – 2015*. Facultad de ingeniería. Universidad Privada de Tacna.

EcuRed. (2012). *IDE de Programación.* Recuperado de https://www.ecured.cu/IDE_de_Programaci%C3%B3n

EcuRed. (2012). *Microsoft SQL Server*. Recuperado de https://www.ecured.cu/Microsoft_SQL_Server

Edwars, C., Ward J. y Bytheway, A. (1998): *Fundamentos de Sistemas de Información*. Madrid, Prentice Hall.

Fergarciac. (2013). *Entorno de Desarrollo Integrado (IDE).* Recuperado de https://fergarciac.wordpress.com/2013/01/25/entorno-de-desarrollo-integrado-ide/

Fontela, A. (2015). *¿Qué es Bootstrap?* Recuperado de https://raiolanetworks.es/blog/que-es-bootstrap/

Franco, A. (2013). *Aplicación web para la administración online de citas médicas en el Centro Médico de Orientación y Planificación Familiar CEMOPLAF-OTAVALO.* Facultad de Ingeniería en Ciencias Aplicadas. Universidad Técnica del Norte, Ibarra Ecuador.

García, F. (s.f.). *Los Sistemas de Información histórica: una nueva frontera en la construcción científica de la historia.* Universidad de Zaragoza.

Gualteros, L. (2016: *Sistema de Información Web para la Gestión de los Recursos Bibliográficos en la Biblioteca de la IED El Rodeo.* Departamento de Facultad Tecnológica. Universidad Distrital Francisco José de Caldas.

Hernández, A. (s.f.). *Los sistemas de información: evolución y desarrollo*. Recuperado de https://dialnet.unirioja.es/servlet/articulo?codigo=793097.

Hernández, H. (2012). L*a importancia de un Sistema de Información en las empresas.* Recuperado de http://hectorhernandezadm.blogspot.mx/

Hernández, U. (2015). *MVC (Model, View, Controller) explicado.* Recuperado de https://codigofacilito.com/articulos/mvc-model-view-controller-explicado

Huayamares, L. (2012). *Implementación de un Sistema de Información para el Área de Registro Civil, en la Municipalidad Distrital de Pueblo Nuevo. F*acultad de Ingeniería, Ciencias y Administración. Universidad Privada Ada a. Byron.

Izamorar. (2017). *Componentes de un Sistema de Información*. Recuperado de https://izamorar.com/componentes-de-un-sistema-de-informacion/

Jack, J. (2014). *Patrón de Diseño MVC (Modelo Vista Controlador) y DAO (Data Access Object).* Recuperado de https://jossjack.wordpress.com/2014/06/22/patron-de-diseno-mvc-modelo-vista-controlador-y-dao-data-access-object/

Jiménez, H. (2013). *Integración de un Sistema Geográfico para el Análisis Inmobiliario.* Departamento de ingeniería Civil y Minas. Universidad de Sonora.

Kniberg, H. (2007). *Scrum y XP desde las Trincheras.* InfoQ. Estados Unidos.

López, M. (2013). *Análisis, Diseño y Desarrollo de un Sistema de Información para soportar el proceso de inventario y préstamos de libros en la biblioteca de la Institución Educativa Alejandro Vélez Barrientos del municipio de Envigado, Antioquia utilizando la plataforma Visual Studio.Net 2010 y SQL Server.* Universidad Nacional Abierta y a Distancia UNAD de Medellín.

Malucín, M. (2015). *Sistema Informático para mejorar la Gestión Comercial del Almacén "Dangelo".* Facultad de Sistemas Mercantiles. Universidad Regional Autónoma de los Andes.

María. (2016). *Qué es Bootstrap y cuáles son sus ventajas.* Recuperado de http://puntoabierto.net/blog/que-es-bootstrap-y-cuales-son-sus-ventajas

Mero, S. (2014). *Sistema Informático de Control de Ejecución de Convenios y Becas de la Unidad de Cooperación Interinstitucional de la Universidad Técnica Estatal de Quevedo.*

Facultad de Ciencias de la Ingeniería Carrera de Ingeniería en Sistemas. Universidad Técnica Estatal de Quevedo.

Microsoft. (2007). *Información general sobre ASP.NET.* Recuperado de https://msdn.microsoft.com/es-es/library/4w3ex9c2(v=vs.100).aspx

Microsoft. (2017). *Información general acerca de .NET Framework.* Recuperado de https://msdn.microsoft.com/es-mx/library/zw4w595w(v=vs.110).aspx

Microsoft. (2017). *Información general de Entity Framework*. Recuperado de https://msdn.microsoft.com/es-es/library/bb399567(v=vs.110).aspx

Microsoft. (2017). *Introducción a Visual Studio.* Recuperado de https://msdn.microsoft.com/es-es/library/fx6bk1f4(v=vs.100).aspx

Microsoft. (2017). *NET Core y código abierto.* Recuperado de https://msdn.microsoft.com/es-es/library/dn878908(v=vs.110).aspx

Monteriro, J. (2001). Qué es CSS. Recuperado de https://desarrolloweb.com/articulos/26.php

Ortega, E. (2012). *Definición y Clasificación de los Sistemas de Información.* Recuperado de http://sisinformacion.obolog.es/definicion-clasificacion-sistema-informacion-2011378

Pavón, J. (s.f.). *Estructura de las Aplicaciones Orientadas a Objetos. El patrón Modelo-Vista-Controlador (MVC).* Departamento de Ingeniería del Software e Inteligencia Artificial, Universidad Complutense Madrid.

Pazmiño, C. (2013). *Análisis, Diseño y Desarrollo de un Sistema de Evaluación de Desempeño Laboral Basado en Competencias*. Facultad de Ingeniería, Ciencias Físicas y Matemática. Universidad Central del Ecuador.

Pérez, D. (2007). *¿Qué es JavaScript?* Recuperado de http://www.maestrosdelweb.com/que-es-javascript

Pinargote, M. y Sánchez, C. (2014). *Sistema Informático de Gestión de Ventas y Servicios Técnicos en la Empresa S-Compu del Cantón Pedernales.* Escuela Superior Politécnica Agropecuaria de Manabí Manuel Félix López.

Ramírez, C. y Vélez, G. (2015). *Automatización del Registro y Control de los Procesos de Hospedaje, Restaurante y Eventos del Hotel-Laboratorio "El Higuerón" de la ESPAM MFL.* Escuela Superior Politécnica Agropecuaria de Manabí Manuel Félix López.

Romero, R. (2012). *Análisis, Diseño e Implementación de un Sistema de Información Aplicado a la Gestión Educativa en Centros de Educación Especial.* Facultad de Ciencias e Ingeniería. Pontificia Universidad Católica del Perú.

Rouse, M. (2017). *SQL Server*. Recuperado de http://searchdatacenter.techtarget.com/es/definicion/SQL-Server

Ruiz, J. (2009). *Sistemas de Gestión de una Inmobiliaria.* Escuela Técnica Superior de Ingeniería (ICAI). Universidad Pontificia Comillas.

Saavedra, J. (2007). *¿Qué es Microsoft.NET?* Recuperado de https://jorgesaavedra.wordpress.com/2007/05/09/%C2%BFque-es-microsoftnet/

Sarmiento, C. (2012). *Sistema de registro de participantes en eventos académicos.* Decanato de Estudios Profesionales Coordinación de Ingeniería de la Computación. Universidad Simón Bolívar.

IX. Anexos

Tipo de documento	**Formato de Protocolo de Proyecto de Prácticas Profesionales**		**Fecha: 01-08-2017**
Área Responsable	**Docencia**		
Nombre del Alumno	**José Escudero Cisneros**	**Número de Cuenta**	**299157**
Nombre del Docente Responsable	**Mtro. Felipe de Jesús Núñez Cárdenas**		
Licenciatura	**Ciencias Computacionales**		
Nombre del Proyecto	**Diseño de Sistemas de Control y Gestión de Proveedores y Agentes para Empresa Inmobiliaria**		

					Agosto				Septiembre				Octubre				Noviembre			
					Semanas															
Porcentaje	Fases				1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
10%	Análisis			P																
				R																
				%	5	5														
20%	Diseño			P																
				R																
				%			5	10	5											
40%	Construcción			P																
				R																
				%						5	5	5	5	5	5	5	5			
20%	Pruebas			P																
				R																
				%														5	5	10
10%	Documentación	No. Act.	Actividad																	
		1	Introducción	P																
				R																

				%	0.5	0.5														
		2	Estado del Arte	P																
				R																
				%		0.5	0.5	0.5												
		3	Marco Teórico	P																
				R																
				%			0.5	0.5	0.5											
		4	Marco Metodológico	P																
				R																
				%					0.5	0.5	0.5									
		5	Desarrollo	P																
				R																
				%								0.5	0.5	1						
		6	Resultados	P																
				R																
				%													0.5	1		
		7	Conclusiones	P																
				R																
				%																
		8	Bibliografías	P															0.5	
				R																
				%																0.25
		9	Anexos	P																
				R																
				%																0.25