TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE ACAPULCO

DESARROLLO DE UNA HERRAMIENTA PARA LA GESTIÓN ADMINISTRATIVA DE SERVIDORES VIRTUALES PRIVADOS PARA UNA EMPRESA DEDICADA A PROVEER SOLUCIONES DE COMUNICACIÓN

TESIS PROFESIONAL

QUE PARA OBTENER EL TÍTULO DE:
MAESTRO EN SISTEMAS COMPUTACIONALES

PRESENTA:
ING. ÁNGEL RODRÍGUEZ RAYO

DIRECTOR DE TESIS:
M.I.D.S. ALMA DELIA DE JESÚS ISLAO

CODIRECTOR DE TESIS:
DR. EDUARDO DE LA CRUZ GÁMEZ

TUTOR DE TESIS:
M.T.I. RAFAEL HERNÁNDEZ REYNA

ACAPULCO, GRO., DICIEMBRE 2020.

El presente trabajo de tesis fue desarrollado en la *División de Estudios de Posgrado e Investigación* del *Instituto Tecnológico de Acapulco*, perteneciente al Programa Nacional de Posgrados de Calidad (PNPC-CONACYT).

Con domicilio para recibir y oír notificaciones en Av. Instituto Tecnológico de Acapulco s/n, Crucero del Cayaco, Acapulco, Guerrero, México. C.P. 39905.

Becario:	Ángel Rodríguez Rayo.
CVU:	928450.
Núm. de apoyo:	711854.
Grado:	Maestría

Descargo de Responsabilidad Institucional

El que suscribe declara que el presente documento titulado “Desarrollo de una herramienta para la gestión administrativa de servidores virtuales privados para una empresa dedicada a proveer soluciones de comunicación” es un trabajo propio y original, el cual no ha sido utilizado anteriormente en institución alguna para propósitos de evaluación, publicación y/o obtención de algún grado académico.

Además, se han recogido todas las fuentes de información utilizadas, las cuales han sido citadas en la sección de referencias bibliográfica de este trabajo.

Acapulco, Gro; a 7 de diciembre de 2020.

Ing. Ángel Rodríguez Rayo

CARTA DE CESIÓN DE DERECHOS DE AUTOR

El que suscribe: Ángel Rodríguez Rayo, autor del trabajo escrito de evaluación profesional en la opción de Tesis Profesional de Maestría con el título "Desarrollo de una herramienta para la gestión administrativa de servidores virtuales privados para una empresa dedicada a proveer soluciones de comunicación", por medio de la presente con fundamento en lo dispuesto en los artículos 5, 18, 24, 25, 27, 30, 32 y 148 de la Ley Federal de Derechos de Autor, así como los numerales 2.15.5 de los lineamientos para la Operación de los Estudios de Posgrado; manifiesto mi autoría y originalidad de la obra mencionada que se presentó en la División de Estudios de Posgrado e Investigación, para ser evaluada con el fin de obtener el Título Profesional de Maestro en Sistemas Computacionales.

Así mismo expreso mi conformidad de ceder los derechos de reproducción, difusión y circulación de esta obra, en forma NO EXCLUSIVA, al Tecnológico Nacional de México campus Acapulco; se podrá realizar a nivel nacional e internacional, de manera parcial o total a través de cualquier medio de información que sea susceptible para ello, en una o varias ocasiones, así como en cualquier soporte documental, todo ello siempre y cuando sus fines sean académicos, humanísticos, tecnológicos, históricos, artísticos, sociales, científicos u otra manifestación de la cultura.

Entendiendo que dicha cesión no genera obligación alguna para el Tecnológico Nacional de México campus Acapulco y que podrá o no ejercer los derechos cedidos. Por lo que el autor da su consentimiento para la publicación de su trabajo escrito de evaluación profesional.

Se firma presente en la ciudad de Acapulco de Juárez, estado de Guerrero a los 7 días del mes de diciembre de 2020.

Ángel Rodríguez Rayo

Acapulco, Gro; a 7 de diciembre de 2020.

AUTORIZACIÓN DE IMPRESIÓN DE TESIS

Los abajo firmantes, miembros de la comisión revisora de tesis designada por la División de Estudios de Posgrado e Investigación del Tecnológico Nacional de México campus Acapulco para la evaluación de la tesis del alumno **Ángel Rodríguez Rayo**, manifiestan que después de haber revisado su tesis: **"Desarrollo de una herramienta para la gestión administrativa de servidores virtuales privados para una empresa dedicada a proveer soluciones de comunicación"** desarrollada bajo la dirección del DIRECTOR, y el CO-DIRECTOR, el trabajo se **ACEPTA** para proceder a su impresión.

A T E N T A M E N T E

M.I.D.S. Alma Delia de Jesús Islao
Cédula Profesional: 8604126

Dr. Eduardo de la Cruz Gámez
Cédula Profesional: 6526073

M.T.I. Rafael Hernández Reyna
Cédula Profesional: 5826794

SECRETARÍA DE EDUCACIÓN PÚBLICA
INSTITUTO TECNOLÓGICO DE ACAPULCO
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

Dr. Eduardo de la Cruz Gámez
Coordinador de la Maestría en Sistemas Computacionales

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA

TECNOLÓGICO NACIONAL DE MÉXICO

Instituto Tecnológico de Acapulco
División de Estudios de Posgrado e Investigación

"2020, Año de Leona Vicario, Benemérita Madre de la Patria"

Acapulco Gro., 8/Diciembre/2020

NO. OFICIO: DEPI-204/2020

ASUNTO:
AUTORIZACIÓN DE
IMPRESIÓN DE TESIS PROFESIONAL

C. ÁNGEL RODRÍGUEZ RAYO

De acuerdo al reglamento de los Institutos Tecnológicos, dependiente de la Secretaría de Educación Pública y habiendo cumplido con todos los requisitos normativos respecto a su trabajo para titulación, Opción Titulación Tesis Profesional, con el proyecto titulado: DESARROLLO DE UNA HERRAMIENTA PARA LA GESTIÓN ADMINISTRATIVA DE SERVIDORES VIRTUALES PRIVADOS PARA UNA EMPRESA DEDICADA A PROVEER SOLUCIONES DE COMINICACIÓN. Se **CONCEDE** la **AUTORIZACIÓN** para que proceda a la impresión del mismo.

Sin otro particular por el momento, me es grato quedar de usted.

A T E N T A M E N T E "
Educación Tecnológica con Compromiso Social"

SECRETARÍA DE EDUCACIÓN PÚBLICA
INSTITUTO TECNOLÓGICO DE ACAPULCO
DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

EDUARDO DE LA CRUZ GÁMEZ
JEFE DE LA DIVISIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

C.c.p. Expediente

EDG/stv

Agradecimientos

Agradezco a Dios por la salud y la vida, por otorgarme la fuerza para seguir adelante.

A mi Directora de Tesis. M.I.D.S. Alma Delia De Jesús Islao, por su valiosa asesoría en el transcurso de la maestría, por alentarme y estar siempre al pendiente de nuestras dudas.

A mi codirector de Tesis y Jefe de Departamento de Posgrado. Dr. Eduardo de la Cruz Gámez, por su consejo y orientación sobre mi proyecto de tesis.

A quienes con su conocimiento, experiencia y guía me formaron como profesional a lo largo de mi carrera académica.

A mis compañeros y amigos de generación, con los que conviví y aprendí a lo largo de este periodo, estoy feliz por haberlos conocido, gracias a quienes siempre me brindaron su apoyo.

A BTU Comunicación por sus atenciones y por permitirme desarrollar el presente proyecto de tesis.

A Conacyt por el apoyo económico a lo largo de dos años en este posgrado y por incentivarme a seguir conociendo y aprendiendo.

A mis padres por inculcarme el hábito del estudio.

Dedicatorias

Dedico esta obra a mis padres que siempre han luchado por brindarle lo mejor a sus hijos, por siempre estar al pendiente de nosotros y cuidarnos en todo momento.

A mi hermano, por estar ahí siempre que lo necesite y brindarme su apoyo incondicional.

A mis compañeros de maestría más cercanos, a quienes considero mis amigos, gracias por brindarme su amistad, Alberto, Alejandro Zapata, Alejandro Bautista, Rogelio y en especial a mi amigo Raúl, quien siempre me apoyo en todas mis dudas y confió en mí.

A mi persona especial que ha estado conmigo desde antes de comenzar esta etapa de mi vida, por motivarme a seguirme preparando, por toda tu paciencia y apoyo incondicional brindado durante todo este tiempo, y por siempre confiar en mí, te quiero Sarai.

Resumen

En el presente trabajo se describe el desarrollo de una herramienta informática que permite gestionar la información relacionada a la contratación de un servicio de servidor virtual y a su vez notificar vía correo electrónico a los clientes cuyo pago se encuentre pendiente.

La herramienta propuesta está diseñada con base en el patrón arquitectónico MVC, mediante el cual está estructurado el marco de desarrollo ASP.NET Core MVC, este patrón consta de tres componentes: modelos: encapsula o representa los datos de la aplicación, el cual es consultado por el controlador y a su vez requerido en la vista para ser representado en ella, vistas: se encarga de la interacción con el usuario y la representación de un modelo mediante una interfaz gráfica de usuario, controlador: es el intermediario entre el modelo y la vista ante las peticiones generadas por el cliente mediante la vista.

Además, la herramienta es una solución distribuida en tres proyectos. Datos donde realizaremos el mapeamiento de datos de nuestra base de datos. Entidades con las clases que van a representar a cada una de las tablas de la base de datos y el proyecto web donde se ha aplicado el patrón MVC, esto para crear una capa de abstracción entre la capa de acceso a datos y la capa de lógica de negocios de la herramienta. Con la implementación de esta división ayuda a aislar la aplicación de los cambios en la persistencia de datos.

La implementación de la herramienta propuesta permitirá fortalecer el departamento de desarrollo y área administrativa de la empresa BTU Comunicación, al proveer una forma de gestionar su información y notificaciones vía correo electrónico a sus clientes de servidores virtuales.

Abstract

This paper describes the development of a computer tool that allows managing the information related to the hiring of a virtual server service and, in turn, notifying customers by email that they are pending to make their corresponding payment.

The proposed tool is designed based on the MVC architectural pattern, through which the ASP.NET Core MVC development framework is structured, this pattern consists of three components: models: encapsulates or represents the application data, which is consulted by the controller and in turn required in the view to be represented in it, views: it is responsible for the interaction with the user and the representation of a model through a graphical user interface, controller: it is the intermediary between the model and the view before the requests generated by the client by sight.

In addition, the tool is a solution distributed in three projects. Data where we will perform data mapping of our database. Entities with the classes that will represent each of the tables in the database and the web project where the MVC pattern has been applied, this to create an abstraction layer between the data access layer and the logic layer Business tool. With the implementation of this division, it helps isolate the application of changes in data persistence.

The implementation of the proposed tool will strengthen the development and administrative area of the BTU Communication company, by providing a way to manage your information and email notifications to your virtual server clients.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1 Introducción

1.1 Antecedentes del Problema

BTU Comunicación es una empresa dedicada a proveer soluciones de comunicación, tanto al ramo empresarial como usuarios particulares. Estas soluciones de comunicación abarcan desde Servicios de Acceso a Internet, como Instalación y configuración de redes LAN y WAN cableadas e inalámbricas, redes privadas virtuales (VPN's), cableado estructurado (UTP y Fibra óptica), soluciones de seguridad (Firewalls, Cámaras de Vigilancia IP, Controles de Acceso) y Servidores Virtuales Privados (VPS).

La compañía tiene el reconocimiento de proveer servicios de muy buena calidad en los diferentes ámbitos que maneja, cuenta con una disponibilidad del 99.73% ya que proporciona un soporte técnico de veinticuatro horas los siete días de la semana. Sin embargo, a lo largo del tiempo que la empresa ha proporcionado estos servicios se ha presentado una problemática, muchos de sus clientes no realizan el pago de su factura en tiempo y forma.

Para la empresa esto genera problemas, principalmente de forma administrativa, y a su vez problemas de credibilidad. Genera problemas de forma económica debido a todos los gastos con los que tiene que cumplir para seguir subsistiendo, renta del inmueble en donde están las oficinas y el pago de sus servicios como lo son luz y agua, enlaces empresariales de Internet, mantenimiento a los aires acondicionados del centro de datos y sistema anti incendios, licencias de Microsoft, Netflow, Monitis, Seguro médico para empleados, de la misma forma realizar los pagos de nómina para los empleados, así como material de trabajo como bobinas de cable UTP de diferentes

categorías, conectores, antenas que trabajan por microondas, Routers, Switches, discos duros, material de papelería y consumibles de oficina.

Por estas razones la empresa al detectar que cierto número de clientes no ha realizado el pago de sus servicios notifica por correo electrónico uno por uno de los clientes de cualquier servicio que está por terminarse el plazo para realizar el pago de su factura, en la gran mayoría de las personas que son notificadas no dan respuesta alguna por lo cual se decide suspender el servicio, ya sea Servidor Virtual Privado, Cuentas de correo, Pago de Hosting Web, Pago de dominio Web, Enlaces de internet dedicados y enlaces domésticos. Para esto hay que identificar qué servicio es el que se le provee a la persona y proceder a suspenderlo.

Esta acción suele generar un impacto negativo en muchos clientes, se comunican al área de soporte para reportar el servicio y argumentando mala calidad, o mencionando porque no se les aviso antes si se iba a realizar algún mantenimiento, diciendo que su servicio no puede estar fuera de línea que es lo que utilizan para trabajar día a día, molestos comentan que no era la primera vez que sucede y se genera un problema mucho más grande de lo que en realidad había sucedido.

A estos clientes se les notifica que han sido suspendidos debido a que no se recibió el pago de su factura, se les comenta que fueron notificados mediante correo electrónico en al menos dos ocasiones en diferentes días al área o persona encargada de realizar los pagos, se les notifica que el periodo para realizar su depósito está por finalizar y de no subsanarse el servicio se verá suspendido.

Esta información es proporcionada principalmente a la persona que opera el servicio que no siempre es la persona que se encarga de realizar el pago, siempre de forma muy respetuosa, e inclusive el cliente sigue molesto afirmando que su departamento, tratándose de una empresa, no es el encargado de realizar los pagos, por lo general quien realiza los pagos es el área de recursos humanos o gestión de pagos. En su gran mayoría solicitan que se les reestablezca el servicio y se comunicarán con el área encargada y realizarán el pago, para esto se solicita primeramente que ellos se comuniquen con su departamento, ya que fue imposible para la empresa contactarlos, también se solicita proporcione fecha y hora que a más tardar se realizará el pago. En algunas ocasiones el cliente vuelve a contactarnos minutos después afirmando que se ha puesto en contacto con el encargado o área que realiza el pago y nos proporcionan la fecha solicitada, se confía en su palabra y se procede a reactivar el servicio para evitar prolongar el conflicto y se espera el pago de acuerdo a lo acordado, en la mayoría de las ocasiones pasa la fecha acordada y no se ha realizado el pago. Por lo cual la empresa se ve en la necesidad de suspender el servicio, regresando a la situación anterior en la que el cliente llama para reportar su servicio, ignorando el hecho de que no se ha subsanado su problema, se vuelve a notificar que no se ha recibido el pago, solicitan que el servicio esta pronto a ser pagado y que se les reestablezca nuevamente el servicio. En esta ocasión ya no se reactiva, se espera se vea reflejado el pago y se procede con la reactivación y notificando mediante correo electrónico.

Este caso suele presentarse en servicios como Servidores Virtuales Privados, Cuentas de correo electrónico, y enlaces de Internet dedicados. Principalmente es una falla de comunicación entre los diferentes departamentos de la compañía cliente los cuales no notifican a otros departamentos

que por tal motivo no se ha realizado el pago de la factura a tiempo, perjudicando a BTU Comunicación.

El problema es más pertinente en el área de Servidores Virtuales Privados y una de las causas que generó este problema es que este ámbito no es el principal servicio que se proporciona, es un área relativamente nueva, misma razón por la cual se les concedieron muchas facilidades a los nuevos clientes, esto con propósito de hacer crecer este sector de productividad, esto provocó que un gran número de clientes cerca del veinte por ciento se despreocuparán en el sentido de realizar los pagos con esta empresa, dejándolos quizá para el final o quitándole prioridad, otros más por el hecho de tener varios servidores rentados con la empresa toman esta actitud, esto no quiere decir que todas las empresas lo hagan de esta forma, hay clientes que tienen más de un servidor y realizan sus pagos en tiempo y forma, me parece que es la razón por la cual la empresa no ha decidido la implementación de un sistema que le permita ser más estricto con sus cobros, sin embargo la cantidad de problemáticas generadas por esta razón son muchas y desgastantes como para los empleados y directivos, además que el recibir de forma atrasada el pago de los servicios genera problemas administrativos para la empresa.

Debido a esto y a que la empresa desea crecer en este sector productivo se ve en la necesidad de adquirir un sistema que administre la disponibilidad delos servicios en relación a su pago para futuros clientes y no dependa de los empleados, se considera que si este problema se repite en varias ocasiones y se toma en cuenta que el número de clientes va en aumento este problema se generaría en una escala mayor y si a los clientes se les notifica de este sistema desde el inicio de la contratación, con el paso del tiempo los resultados se verían reflejados, la idea es recibir el pago

en tiempo y forma y evitar confrontaciones por el funcionamiento del servicio y pérdida de ingresos.

De esta forma los clientes no prolongarían más la fecha para realizar su pago y sabrían que no existe otra forma para restablecer su servicio que liquidando su factura, ya no tratarían de llamar a la línea de soporte para restablecer su servicio o con los administrativos para que se los restablezca, debido a que ninguno de ellos tiene un control de eso y de esta forma evitar la prolongación de pagos actual.

1.2 Planteamiento del problema

En BTU Comunicación actualmente no se cuenta con un sistema que controle la disposición de sus servicios mediante la realización de un pago en tiempo y forma, el no contar con esta plataforma ha concedido la oportunidad para que se generen malos entendidos entre los clientes y la compañía, así como problemas administrativos dentro de la misma, viéndose obligada a utilizar recursos de otras áreas para subsanar otros gastos.

Es importante atender este problema ya que es de gran importancia para la empresa, porque la generación de este tipo de conflictos puede repercutir para el futuro, ya sea que el cliente busque otro proveedor, no contrate más servicios con la empresa o simplemente siga prolongando sus pagos. Para la empresa es un tema de alta relevancia el cuidar su imagen y no perder clientes, ya que esto no atrae a nuevos posibles clientes.

La forma en que afecta este problema a BTU Comunicación, aparte del tema de generar mala imagen con otros posibles clientes, puede causar molestias a los empleados de la empresa, esto repercute en diferentes causas, puede generar que las personas que atienden el área de soporte técnico lleguen a molestarse, desencadenándose en una posible mala atención a otros clientes que no tienen relación alguna, dando una mala impresión, puede generar un ambiente denso y provocar que los empleados no se sientan cómodos con su trabajo. Esto si no se puede manejar y se deja crecer puede ocasionar pérdida de empleados, lo que ocasiona pérdidas para la empresa, ya que se tiene que solicitar nuevamente personal para esta área, y esto lleva un proceso que genera gastos de recursos económicas para la empresa o sobre carga de trabajo para algunos empleados, lo cual, de la misma forma podría genera inconformidades. Las entrevistas y capacitación del nuevo

personal consumen mucho tiempo, el cual podría ser utilizado para realizar alguna otra actividad de mayor provecho para la empresa.

El problema en esta empresa es que no se incorporó desde un inicio este software y no se generó esta norma desde un comienzo, probablemente por el hecho de que no es una empresa muy grande o con renombre como lo es Telmex o Amazon, que cortan de inmediato el servicio al no recibir el pago acordado, el principal objetivo fue adquirir clientes y hacer crecer esta infraestructura, aún no es una infraestructura tan grande como la de otras compañías y se empiezan a generar este tipo de problemas, la razón por la cual se busca dar solución a este problema.

La empresa BTU Comunicación provee una gran cantidad de servicios ya mencionados, la orientación de este sistema va dirigido a los Servidores Virtuales Privados, ya que de todas las áreas de trabajo es en la cual más se presenta y genera un mayor impacto para la empresa, ya que la mayoría de las empresas almacena en ellos las aplicaciones que utilizan cotidianamente para laborar. Esperando que con la implementación de este sistema el problema pueda verse subsanado en su mayoría.

El problema identificado comienza desde el envío por operador de notificaciones por medio de correo electrónico, las cuales se envían faltando cinco y dos días para concluir con el límite del periodo para realizar su pago, suspender el servicio de forma manual, en donde el personal administrativo se comunica con el área de soporte técnico cada vez que desea suspender un servicio, notificación de que el equipo ha sido suspendido por falta de pago, y volver a notificar cuando el servicio ha sido reactivado.

1.3 Objetivos

1.3.1 Objetivo general

Desarrollar una herramienta que notifique de forma automática vía correo electrónico a los clientes que esté por expirar su periodo de pago por el servicio de servidor virtual privado en BTU Comunicación.

1.3.2 Objetivos específicos

- Desarrollar una herramienta que automatice el envío de notificaciones vía correo electrónico a los clientes faltando diez, cinco y dos días antes de terminar su periodo.
- Desarrollar una herramienta que permita el control (Registro, Modificación, Búsqueda y Suspensión) de clientes, servidores y consumibles de servidores virtuales privados dentro de la organización.
- Llevar un control de las notificaciones relacionadas con los Servidores.

1.4 Hipótesis

La implementación de una herramienta para la gestión administrativa de servidores virtuales agilizará el proceso de notificar mediante correo electrónico a los clientes que aún no hayan proporcionado su pago correspondiente. De esta forma la empresa BTU Comunicación obtendrá un control de notificaciones, evidencia que puede ser utilizada de su parte en caso de presentarse una aclaración de disponibilidad de servicio por parte del cliente. Permitiendo aprovechar de una manera más eficiente el recurso humano en actividades de mayor contribución para la organización.

1.5 Justificación

Este trabajo de tesis propone desarrollar una herramienta que permita gestionar al área administrativa de la empresa BTU Comunicación la notificación vía correo electrónico a sus clientes cuyo periodo de pago esté por expirar.

La herramienta que se pretende desarrollar permitirá disminuir la cantidad de pasos a llevar acabo para notificar por correo electrónico a un propietario de un servicio de servidor virtual (VPS).

Por lo tanto la implementación de esta herramienta podría ayudar a disminuir la cantidad de clientes renuentes y agilizar el proceso de notificación mediante correo electrónico, debido a que se ha detectado el atraso de los pagos como un problema para la empresa ya que el personal administrativo invierte en ocasiones una gran cantidad de tiempo en la gestión de estas actividades, por lo cual se ha decidido implementar una herramienta que ayude a minimizar este problema, reduciendo procedimientos repetitivos para el personal y mejorando el rendimiento de la organización.

Impacto tecnológico. Puede causar un impacto tecnológico debido a que es una solución desarrollada y adaptada a las necesidades de BTU Comunicación, por lo que por consecuencia ayudará al crecimiento de las necesidades de automatización de los otros servicios que proporciona.

1.6 Alcances y Limitaciones

1.6.1 Alcances

- La herramienta permitirá el envío de notificaciones por correo electrónico a los clientes que se encuentren pendientes de realizar su pago.
- La herramienta llevará un control de notificaciones emitidas, funcionando como una evidencia en caso de presentarse una aclaración de disponibilidad por parte del cliente.
- La implementación de este sistema generará evidencia de los diferentes avisos que se le realizaron al cliente antes de suspender su Servidor Virtual.

1.6.2 Delimitaciones

- El uso de este sistema no garantizará que los clientes realicen el pago de su servicio de servidores virtuales privados en tiempo y forma.
- Este sistema no realizará el cobro, ni la generación de la factura de los servidores virtuales privados.

Capítulo 2 Consideraciones Teóricas

2.1 Estado del arte

Título: "Virtualización de Redes y Servidores Emulando Infraestructuras Tecnológicas"

Autores: "Juan Casierra Cavada, Xavier Quiñónez Ku, Luis Herrera Izquierdo y Carlos Egas Acosta"

Fecha: 13 de marzo de 2018.

Publicado en: Rev. Hallazgos21, Vol. 3, Suplemento Especial, 2018.

Síntesis:

En este artículo se trata la necesidad de desarrollar destrezas en el área de tecnologías de la información y comunicación. También el crecimiento de requerimientos en tecnologías de virtualización aplicadas a centros de datos, gestionando plataformas tales como **Xen Server**, Vmware, Proxmox, entre otros. Se plantea el objetivo de formular escenarios tecnológicos que permitan la ejecución de prácticas profesionales en ambientes virtuales y, así, mantener actualizados conceptos para afrontar los desafíos de la industria tecnológica.

La virtualización se llevó a cabo dentro de la Escuela de Sistemas y Computación de la Pontificia Universidad Católica del Ecuador, Sede Esmeraldas.

Resultados:

El aporte relevante del proyecto es la utilización de Citrix Xen Server como plataforma de virtualización entre las anteriormente mencionadas.

Los resultados obtenidos demostraron la efectividad de los procedimientos virtuales de forma considerable, permitiendo la gestión de comunicaciones, compartición y balanceo de recursos de hardware, en base a requerimientos; lo que se convierte en una opción fundamental en el ámbito de redes de comunicaciones empresariales, con algunas mínimas limitaciones de compatibilidad.

Dentro del Centro de Datos se desarrolló el diseño de una zona desmilitarizada utilizando un equipo existente, usando **Citrix Xen Server** como plataforma central de virtualización.

Conclusión personal:

Al realizar la virtualización de una infraestructura de tecnologías, y romper los paradigmas de implementación de los procesos tradicionales, como lo evidenció este artículo, permite la continuidad del funcionamiento de la infraestructura de tecnologías, dada su facilidad de redundancia y tiempos de recuperación.

Además de generar un mayor ahorro en el mantenimiento de su infraestructura, se sumaron costos directos e indirectos anuales, se generó un ahorro que permitiría invertir en un mayor ancho de banda, e implementaciones intranet dentro de la institución donde se implementó la virtualización.

La relación que tiene mi trabajo con este artículo es la utilización del sistema de virtualización Citrix Xen Server, que es un producto que no requiere el pago de una licencia para trabajar, que es el Sistema en el cual se virtualizará el Servidor Virtual Privado con el que voy a trabajar y uno de los sistemas con los cuales la empresa realiza la virtualización de sus servidores al igual que VMWare.

Título: “Simulando Proyectos de Desarrollo de Software Administrados con Scrum”

Autor(es): “Diego Alberto Godoy, Edgardo A. Belloni, Henry Kotynski, Hector Dos Santos, Eduardo O. Sosa”

Fecha: 2014

Publicado en: WICC 2014 XVI Workshop de Investigadores en Ciencias de la Computación.

Síntesis:

En este artículo se presenta una de investigación que tiene como objetivo principal la elaboración e implementación de un modelo que simule la metodología de gestión de proyectos *Scrum*.

El trabajo presentado en este artículo tiene como contexto marco el proyecto de investigación denominado “Simulación como Herramienta para la mejora de procesos de Software”, registrado actualmente en la secretaria de investigación y desarrollo de la Universidad Gaston Dachary (UGD) y radicado en el centro de Investigación en Tecnologías de la Información y Comunicaciones de dicha universidad.

Resultados:

En este artículo se presentaron avances de la elaboración de un “Modelo de Simulación Dinámico de Gestión de Proyectos de Desarrollo de Software administrados con *Scrum*”. Al

examinar en las características de la metodología *Scrum*, este documento se des asemeja de otros en los que se diseñan sistemas con las metodologías tradicionales es que la estructura permite la modificación de los datos durante su ejecución.

Conclusión:

Los autores concluyeron en este artículo que la validación realizada con datos tomados de los proyectos y como situaciones de practica se califica como positiva ya que el diseño elaborado se desarrolló de acuerdo con la información de proyectos reales.

La relación que tiene mi trabajo con este artículo es la implementación de la Metodología *Scrum* para el desarrollo de software, ya que en este artículo se plantea la complejidad de la metodología y las ventajas que otorga y las desventajas de utilizar metodologías tradicionales, como lo son la evaluación de riesgos, que es muy compleja y la excesiva flexibilidad para algunos proyectos.

Título: “Laboratorio Virtual y Remoto para la Enseñanza de Diseño y Administración de Redes de Computadoras”

Autor(es): “Daniel Britos, Laura Vargas, Silvia Arias, Nicolás Giraudo, Guillermo Veneranda”

Fecha: 2013

Publicado en: XV WORKSHOP De Investigadores en Ciencia de la Computación

Síntesis:

En este artículo se relata cómo fue desarrollado un laboratorio virtual para los alumnos de la especialidad de ingeniería, debido a que las asignaturas relacionadas con redes de computadoras están orientadas a que el alumno alcance competencias de diseño, administración de redes e implementación.

Debido al alto costo que conlleva laborar en laboratorios con equipos físicos y considerando que en administración de redes se suele realizar en forma remota, se decidió implementar un laboratorio virtual de redes de computo.

Resultados:

El aporte relevante de este artículo es la arquitectura propuesta para el desarrollo de su laboratorio virtual, en base al funcionamiento en conjunto, es la manera más ágil de laborar con estas aplicaciones es ejecutándolas desde el mismo computador.

La virtualización de servidores requiere una capacidad muy grande de caracteristicas computacionales. Separar la carga del procesamiento requiere que la GNS3 pueda comunicarser con una entidad aplicativa de VirtualBox ejecutandose en otro sistema. La transferencia se logra desplegando VBoxWrapper en el servicio de virtualización y ajustando GNS3 con el direccionamiento IP de sí mismo. Una vez hecho realizado, será capaz de encender y apagar los VPS a distancia con GNS3.

Conclusión:

Los autores concluyen que Las aplicaciones GNS3 y VirtualBox nos ayudan a generar topologías completas de redes de ordenadores excluyendo la necesidad de poseer equipos de interconexión mucho menos una cantidad muy amplia de recursos de procesamiento.

La relación que tiene mi trabajo con este artículo es cuando se menciona que podemos configurar GNS3 con la dirección IP de los Servidores Virtuales, comenta que una vez logrado, es capaz de encender y apagar los Servidores Virtuales remotamente usando GNS3, esto es relevante para mi proyecto ya que una de las funciones que realizará es el control de acceso de un servidor virtual, y una de las opciones para realizar esta operación es apagarlo.

Título: "Desarrollo de una aplicación para la enseñanza del idioma *Kichwa* utilizando conceptos web 2.0 y herramientas libres"

Autor: "Wilson Fabián Chiza Morán"

Fecha: 15/dic/2017

Publicado en: Repositorio Digital – Universidad Técnica del Norte

Síntesis:

El presente proyecto de grado es el "Elaboración de una herramienta para el aprendizaje del lenguaje *Kichwa* con ayuda de conceptos Web" guiándose en la metodología de desarrollo ágil *Extreme Programming* (XP) para la culminación del proyecto. Como primer punto, se declara los objetivos del proyecto, se realiza una breve descripción de los antecedentes del *Kichwa*, y se definen los módulos que abarcará la aplicación con sus respectivos contenidos. Posteriormente, se describe algunos conceptos relacionados al Kichwa utilizando datos de INEC también se describe las herramientas y tecnologías que se utilizarán en el presente proyecto. Continuando con el contenido, se documenta las etapas que se realizaron obteniendo los diferentes artefactos que surgen al emplear la metodología de desarrollo XP. Siguiendo con el contenido del artículo, se realiza una descripción más detallada del funcionamiento del aplicativo. Por último, se presenta las conclusiones y recomendaciones que surgieron en el desarrollo del proyecto.

Resultados:

El aporte relevante de este artículo son las herramientas que fueron empleadas para el desarrollo de la aplicación, el lenguaje de programación php, YII Framework, base de datos *MariaDB* y la arquitectura MVC.

Una vez culminado con todas las fases de la metodología y terminado de desarrollar el aplicativo, se procedió a las pruebas de funcionalidad y se obtuvieron resultados satisfactorios en cada uno de los módulos de la aplicación.

Conclusión:

El autor concluye mencionando que las herramientas libres son buenas alternativas para utilizarlos en proyectos similares ya que el costo de desarrollo no se vería incrementado.

El empleo de Frameworks, especialmente los que fueron creados utilizando la arquitectura MVC, en todo tipo de proyectos ayudan significativamente al avance y rápido desarrollo de los aplicativos.

La arquitectura MVC (Modelo Vista Controlador) en una opción al momento de desarrollar software y la cual ayuda a tener un proyecto ordenado para posteriores mantenimientos.

La relación que tiene mi trabajo con este artículo es principalmente la utilización de la arquitectura MVC para el desarrollo de su aplicación web, esto quiere decir que MVC es un patrón de diseño implementado en la elaboración de software. Su principal propósito es crear diferencias entre la forma en el que la aplicación manipula a información.

Título: "Tecnologías de virtualización en los sistemas informáticos de las organizaciones empresariales del estado Zulia"

Autor: "Lugo Cardozo, Neury"

Fecha: 28/05/2014

Publicado en: Universidad Privada Dr. Rafael Belloso Chacín

Síntesis:

En este artículo se realizó una investigación, la cual tiene como razón principal estudiar las diferentes tecnologías utilizadas para la virtualización en los ambientes computacionales de las empresas de Zulia, especializando la virtualización de máquinas de almacenamiento, de redes y de lugares de trabajo.

Resultados:

El aporte relevante de este artículo es los diferentes tipos de virtualización que se producen de forma empresarial y sus clasificaciones, como lo es la virtualización de sistemas operativos, la virtualización completa. De igual forma la virtualización de almacenamiento, la virtualización de redes y la virtualización de estaciones de trabajo.

Conclusión:

Se concluyo que las tecnologías de virtualización utilizadas en los sistemas nos generan beneficios muy relevantes y resultan ser un alto nivel dentro de sus instalaciones de TI; para la virtualización de máquinas, datos, conexiones y para estaciones de trabajo.

La relación que tiene mi trabajo con este artículo es la investigación realizada, menciona que la mayoría de las empresas está optando por elegir servidores virtuales además de otros servicios virtualizados, menciona las diferentes tecnologías que se utilizan para realizar la virtualización como lo es VMWare, uno de los sistemas de virtualización con los que proporciona servicios la empresa donde se establecerá el sistema.

Título: “Estudio comparativo entre las metodologías ágiles y las metodologías tradicionales para la gestión de proyectos software”

Autor: “Manuel José García Rodríguez”

Fecha: Julio 2015

Publicado en: Universidad de Oviedo

Síntesis:

Este documento de tesis tiene como fin introducirse dentro de la investigación de los métodos de administración de herramientas de tipo web. Tienen un conjunto de cualidades que los convierten en diferentes a los demás sistemas de software.

Además de incluir una breve entrada sobre lo que es la ingeniería de software, la administración de sistemas y las razones de sus caídas.

Resultados:

El aporte relevante de este documento son las diferencias encontradas entre las metodologías agiles que son muy heterogéneas entre si contra las metodologías tradicionales.

Además de estudiar tres diferentes parámetros como lo son, costes de los cambios, retorno de la inversión y áreas de conocimiento.

Las metodologías analizadas fueron, *PMBOK*, *PRINCE2*, *ICB*, *XP*, *SCRUM y FDD*, en las áreas de integración, involucrados, alcance, plazos, coste, calidad, recursos humanos, comunicaciones, riesgos y aprovisionamiento.

Conclusión:

El autor concluye en este documento de tesis que se ha conseguido realizar una presentación general de la ingeniería de software, las metodologías de desarrollo de software que hay en la actualidad y una comparación ilustrativa entre ellas.

Además, concluye que se han encontrado dificultades para realizar la comparación entre distintas metodologías. Esto debido a que no es trivial encontrar elementos comunes a todas las metodologías y que, además, las definan completamente.

La relación que tiene mi trabajo con este artículo es que abarca ambos tipos de metodologías, tradicionales como ágiles, esto me ayuda a definir en base a las tablas de resultados que tipo de metodología es la más adecuada para el proyecto que estoy desarrollando.

2.2 Metodologías

2.2.1 SCRUM

Se propone la metodología SCRUM para el desarrollo de este proyecto, debido a que será dividido en una serie de módulos, los cuales deberán analizarse y diseñarse para dar paso a las pruebas unitarias, tanto en el lugar de desarrollo como también frente a los involucrados de la creación del proyecto, lo cual da una ventaja en el aspecto de presentar las diferentes vistas de las que estará compuesta la aplicación web. Esto sumado a la retroalimentación que existirá durante las presentaciones o en caso de que se presenten requerimientos cambiantes o poco definidos, lo cual es una de las características principales de esta metodología.

SCRUM es una metodología de desarrollo ágil, basada en *Sprints*: estos son iteraciones o ciclos de desarrollo cortos en los que se diseña y desarrolla un incremento del sistema. SCRUM puede utilizarse como fundamento de administración de esquemas ágiles, y puede trabajar en conjunto con otras metodologías (Schwaber & Sutherland, 2013).

Esta metodología se secciona en tres fases de trabajo (Sommerville, Administración de un proyecto ágil, 2011). En la primera se realiza la planeación del bosquejo y diseño de la arquitectura, se establecen los objetivos del sistema y el alcance de este. En la segunda fase se realiza una serie de iteraciones o ciclos sprint en los que se establece una lista de requerimientos llamada *product backlog*, y posterior a esto se analiza, diseña y desarrolla cada uno de los módulos de los que estará conformado el sistema. Cabe mencionar que el cliente o usuario final al que pertenecerá el sistema interviene en esta fase con la finalidad de añadir nuevos requerimientos o tareas y revisar los ya desarrollados de forma que exista una retroalimentación entre desarrolladores y usuarios finales y pulir el cierre del sprint.

En la última fase se concluye el proyecto, se compacta toda la documentación generada, esto incluye manuales de usuario del sistema, manuales técnicos del sistema, documentación de apoyo, y una vez completado se entrega el sistema y sus componentes a los usuarios finales o clientes.

Figura 2.1 Fases de un sprint en la metodología SCRUM (Sommerville, 2011).

Como se redacta al principio, un *sprint* de SCRUM es una iteración en la que se reciben los requerimientos totales del cliente/usuarios finales de los cuales se seleccionan los requerimientos a realizar a través de un análisis de prioridades, se generan los diagramas y la documentación necesaria para dar paso al desarrollo del o de los módulos propuestos en el *sprint*. Al finalizar la iteración, se revisa la documentación, se presenta y se entrega la funcionalidad completa a los participantes y/o usuarios finales. Cada una de estas iteraciones tiene una duración determinada, por lo general de 2 a 6 semanas.

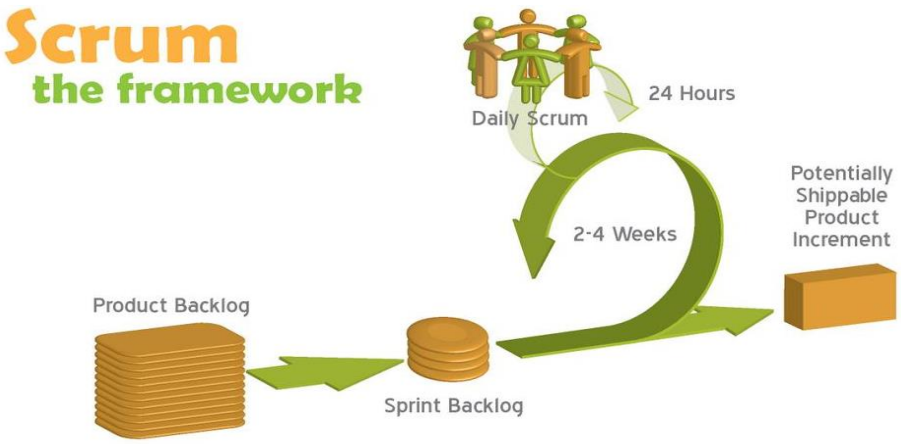

Figura 2.2 Proceso de actividades que se desarrollan en un ciclo (Carrión Abollaneda, 2015).

2.2.2 RUP (Proceso Unificado de Rational)

Proporciona un enfoque disciplinado para la asignación de tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es garantizar la producción de alta calidad software que satisfaga las necesidades de sus usuarios finales, dentro de un horario predecible y presupuesto.

Características esenciales

Este proceso de software RUP tiene tres características esenciales: Dirigido por los Casos de Uso, Centrado en la arquitectura, y es iterativo e incremental.

1. Proceso dirigido por Casos de Uso: En RUP los Casos de Uso no son sólo una herramienta para especificar los requisitos del sistema. También guían su diseño, implementación y prueba.

2. Proceso centrado en la arquitectura: En el caso de RUP además de utilizar los Casos de Uso para guiar el proceso se presta especial atención al establecimiento temprano de una buena arquitectura que no se vea fuertemente impactada ante cambios posteriores durante la construcción y el mantenimiento.

3. Proceso iterativo e incremental: La estrategia que se propone en RUP es tener un proceso iterativo e incremental en donde el trabajo se divide en partes más pequeñas o mini proyectos. Permitiendo que el equilibrio entre Casos de Uso y arquitectura se vaya logrando durante cada mini proyecto, así durante todo el proceso de desarrollo. Cada mini proyecto se puede ver como una iteración, un recorrido más o menos completo a lo largo de todos los flujos de trabajo fundamentales, del cual se obtiene un incremento que produce un crecimiento en el producto.

Los requerimientos se dividen en dos grupos. Los requerimientos funcionales representan la funcionalidad del sistema. Se modelan mediante diagramas de Casos de Uso. Los

requerimientos no funcionales representan aquellos atributos que debe exhibir el sistema, pero que no son una funcionalidad específica. Por ejemplo, requisitos de facilidad de uso, fiabilidad, eficiencia, portabilidad. (Sommerville, Ingeniería de Software, 2011)

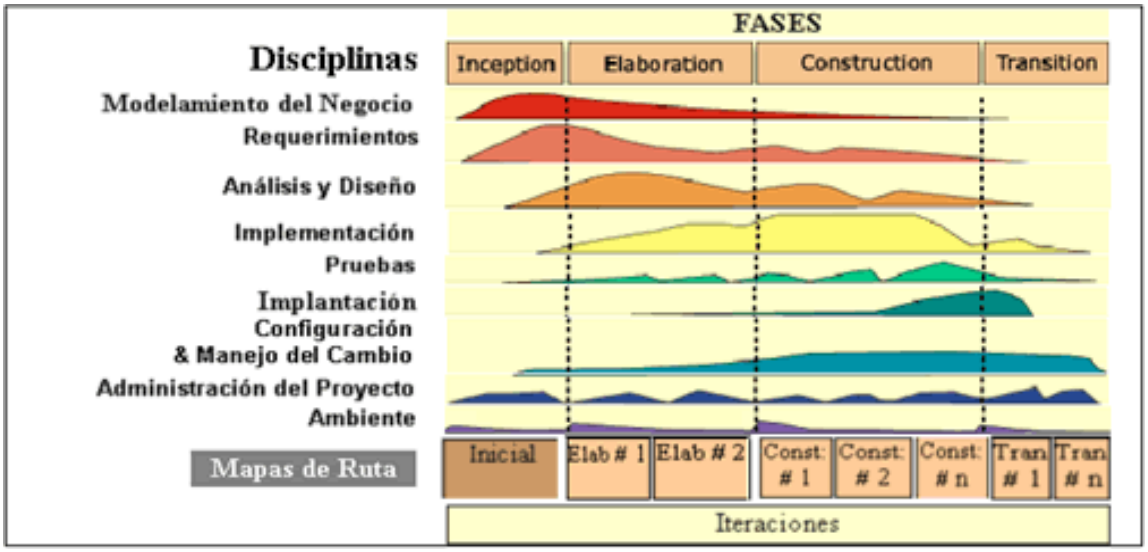

Figura 2.3 Fases RUP (Sommerville, 2011).

2.3 Lenguaje Unificado de Modelado (UML)

El modelado de sistemas nos es útil para desarrollar diagramas de modelado abstractos de una herramienta, donde cada entidad presenta un punto de vista o perspectiva diferente de dichos sistemas. En realidad, la modelación de aplicaciones ha venido desarrollándose en un ambiente para demostrar el funcionamiento utilizando cierta forma de notación visual, que por lo general suele basarse en notaciones del UML (Sommerville, 2016).

El UML tiene diversas formas de representar el modelado y, por consecuente, sostiene el desarrollo de diversos tipos de modelos de sistemas. En realidad, una investigación en 2007 (Siau y Erickson, 2007) demostró que un gran porcentaje de usuarios del UML consideraban que cinco tipos de diagrama podrían describir lo más importante de una herramienta (Sommerville, 2016).

1. Diagramas de actividad, expresan las tareas principales en el desarrollo o en la administración de información.

2. Diagramas de casos de uso, muestran diferentes actividades en medio de una herramienta y su ambiente.

3. Diagramas de secuencias, exponen los procedimientos en medio de los actores y la aplicación, y en medio de los objetos de la herramienta.

4. Diagramas de clase, muestran las clases de un componente en la herramienta y las relaciones en medio de sus diversas clases.

5. Diagramas de estado, muestran la interacción de la herramienta delante las actividades internas y externas.

A continuación, se fundamentan dos tipos de diagramas que serán implementados para modelar la aplicación propuesta en el objetivo de la tesis.

2.3.1 Casos de uso

El diseño de casos de uso suele ser utilizado en gran parte para ayudar en la obtención de requerimientos. Un caso de uso es posible tomarlo como un sencillo ambiente que interprete lo que esperará el cliente de una herramienta.

En su apariencia más sencilla, los casos de uso lo interpretamos como un ovalo, entre los actores que participan dentro del caso de uso demostrados como imágenes de personas. La figura 2.4 demuestra un caso de uso de una herramienta de gestión de pacientes – cuidado a su estado psicoanalítico; que incluye la tarea de almacenar información desde este sistema a una aplicación sumamente completa de registro de pacientes.

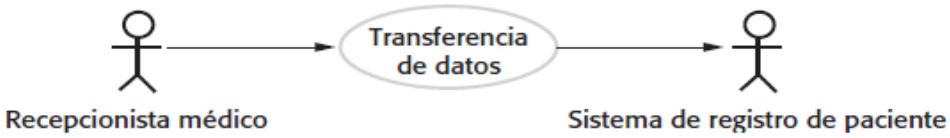

Figura 2.4 Caso de uso de transferencia de datos (Sommerville, 2016).

Observe que en este caso de uso hay dos actores: el operador que transfiere los datos y el sistema de registro de pacientes. La notación con figura humana se desarrolló originalmente para cubrir la interacción entre individuos, pero también se usa ahora para representar otros sistemas externos y el hardware. De manera formal, los diagramas de caso de uso deben emplear líneas sin flechas; las flechas en el UML indican dirección del flujo de mensajes. Evidentemente, en un caso de uso los mensajes pasan en ambas direcciones. Sin embargo, las flechas en la ilustración 2.4 se usan de manera informal para indicar que el recepcionista médico inicia la transacción y los datos se transfieren al sistema de registro de pacientes.

Los diagramas de caso de uso brindan un panorama sencillo de una interacción, de modo que se tiene que ofrecer más detalle para entender lo que está implicado, el formato con más relevancia es el formato tabular estándar. En la ilustración 2.5 se muestra una interpretación tabular del caso de uso “envió de información” (Sommerville, Ingenieria de Software, 2016).

Sistema de administración de pacientes-atención a la salud mental	
Actores	Recepcionista médico, sistema de registro de pacientes.
Descripción	Un recepcionista puede transferir datos del sistema de administración de paciente-atención a la salud mental a una base de datos general de registros de pacientes, mantenida por una autoridad sanitaria. La información transferida puede ser información personal actualizada o un resumen del diagnóstico y tratamiento del paciente.
Datos	Información personal del paciente, resumen de tratamiento.
Estímulo	Comando de usuario emitido por recepcionista médico.
Respuesta	Confirmación de que el sistema de registro de pacientes se actualizó.
Comentarios	El recepcionista debe tener permisos de seguridad adecuados para acceder a la información del paciente y al sistema de registro de pacientes.

Figura 2.5 Descripción del caso de uso “trasmisión de información” (Sommerville, 2016).

En la ilustración 2.6 se muestra el caso de uso de las acciones que puede hacer el recepcionista médico con el módulo de pacientes tales como: registrar, dar de baja, ver información, transferir datos y contacto del paciente.

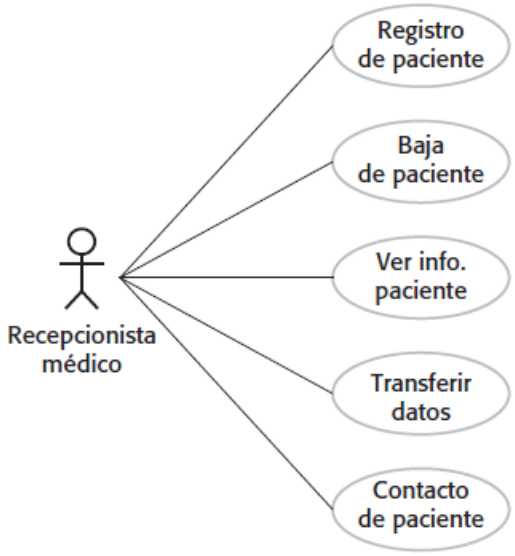

Figura 2.6 Casos de uso que incluyen el papel "recepcionistaMédico" (Sommerville, 2016).

Tabla 2.1 Tipos de relaciones de casos de uso (Rumbaugh, Jacobson, & Booch, 2000).

Relación	*Función*	*Notación*
asociación	La línea de comunicación entre un actor y un caso de uso en el que participa	———
extensión	La inserción de comportamiento adicional en un caso de uso base que no tiene conocimiento sobre él	«**extend**» - - - >
generalización de casos de uso	Una relación entre un caso de uso general y un caso de uso más específico, que hereda y añade propiedades a aquél	——▷
inclusión	Inserción de comportamiento adicional en un caso de uso base, que describe explícitamente la inserción	«**include**» - - - >

Las relaciones de inclusión y extensión pueden ser graficadas como fechas de líneas discontinuas usando la palabra clave include o extend como se muestra en la ilustración 2.8. La

relación de inclusión señala al caso de uso a ser incluido; la relación de extensión señala el caso de uso que se expandirá.

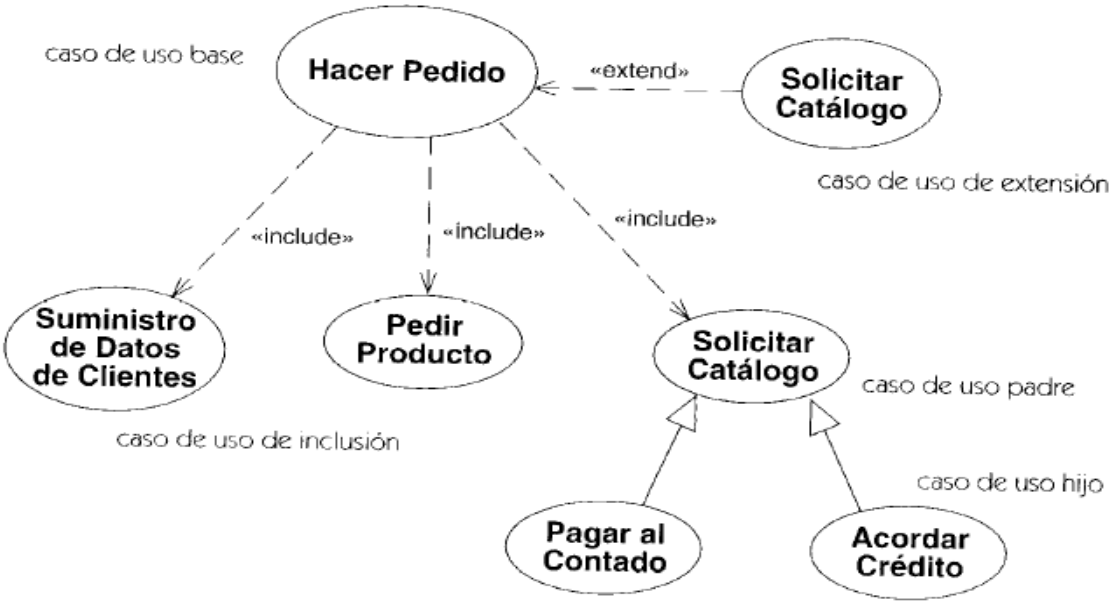

Figura 2.7 Relaciones de casos de uso (Rumbaugh, Jacobson, & Booch, 2000).

2.3.2 Diagrama de clase

Los diagramas de clases suelen implementarse cuando se diseña el modelado de una herramienta web orientada a objetos para representar las clases de una aplicación y sus respectivas líneas de asociación entre clases. Dicho de otra forma, una clase de objeto es tratada como la expresión general de un tipo de objeto de la aplicación. Una asociación es la forma de en qué se pueden relacionar las clases entre sí. En conclusión, cada clase puede tener el conocimiento de la clase vinculada.

Al desarrollar modelos es importante que, mientras las etapas principales del diseño de ingeniería de software, los objetos expresen una entidad de nuestro entorno actual, como un automóvil, una motocicleta, un mecánico, etcétera. Mediante se realiza la implementación de un desarrollo, por lo general es necesario dar a conocer los componentes adicionales de despliegue

que usaran para proporcionar cierta ocupación o la funcionalidad indispensable de una herramienta digital.

En esta situación, la razón se encuentra en el modelado de objetos reales, como requerimientos o los procesos principales del modelado de sistema.

Los diagramas de clase UML es posible expresarlos con diferentes grados de detalle. Al momento de elaborar una aplicación, la principal parte con frecuencia se basa en detectar los componentes principales y modelarlos como clases. Una de las formas más comunes de hacerlo es redactar el identificador de la clase en un cuadro. (Sommerville, Ingenieria de Software, 2016).

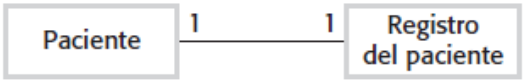

Figura 2.8 Clases y asignación UML. (Sommerville, 2016).

Como ejemplo, la figura 2.8 es un diagrama de clase sencillo que interpreta dos clases: Paciente y Registro del paciente, con una asignación entre ambos.

En la figura 2.8 se muestra un detalle más de los diagramas de clase: la habilidad para identificar cuántos componentes interviene en la relación. Durante el ejemplo, cada final de la asociación se identifica con un 1, lo cual da a entender que existe una asociación 1:1 entre los componentes de estas clases. Da a entender que, cada paciente tiene precisamente un registro, y cada registro almacena datos precisos del cliente. En el siguiente diagrama se muestra que son capaces otras multiplicidades. Se definen un número preciso de componentes que están relacionados, o en efecto, con la implementación de un asterisco (*), como se observa en la figura 2.9, donde existe un número indeterminado de componentes en la relación.

En la figura 2.9 se diseña usualmente el diagrama de clase para identificar a los componentes de la clase “paciente” de igual forma colaboran en las asociaciones con varias otras clases. De esta

forma, el UML admite especificar la representación de los componentes que intervienen en la relación (Sommerville, 2016).

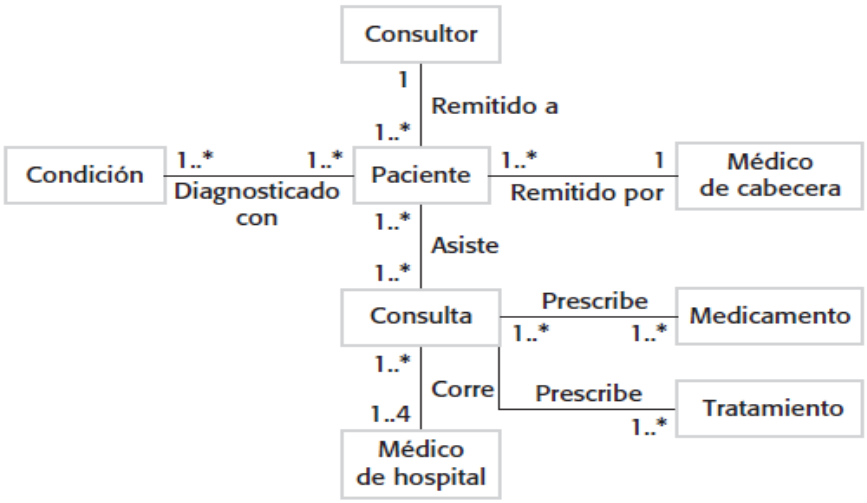

Figura 2.9 Clases y asociaciones en la herramienta de administración de clientes - cuidado a la integridad psicoanalítica (Sommerville, 2016).

Durante esta sección de especificación, los diagramas de clase aparentan ser diseños semánticos de información. Los modelados semánticos de información se utilizan para el modelado de entornos de información. Proporcionan entidades de datos, sus características específicas y sus asociaciones en medio de dichas tablas. El UML no proporciona una regulación escrita detallada para este diseño de modelado de arreglo de datos, debido a que considera el desarrollo orientado a componentes u objetos, de esta forma los diagramas de información que utilizan componentes y sus asociaciones. sin embargo, es capaz de usar el UML para exponer un diagrama semántico de información. Esto se muestra en la figura 2.10, donde:

1. El nombre de la clase de objeto está en la sección superior.
2. Los atributos de la clase están en la sección media. Esto debe incluir los nombres del atributo y, opcionalmente sus tipos.

3. Las operaciones (llamadas métodos en los lenguajes de programación) asociadas con la clase de objeto, están en la sección inferior del rectángulo (Sommerville, Ingenieria de Software, 2016).

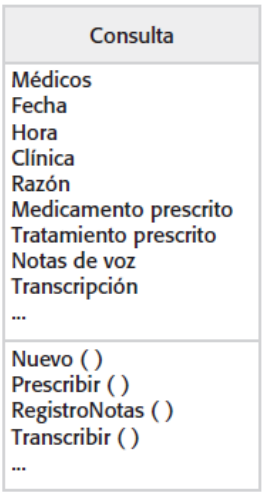

Figura 2.10 La clase de consulta (Sommerville, 2016).

Una vez descritos algunos de los tipos de diagramas a emplear en este trabajo, es recomendable tener en consideración que en el desarrollo de software existe una separación en el **front-end** y **back-end** lo cual es un tipo de abstracción que ayuda mantener separada las diferentes tecnologías y partes de un software. A continuación, se describen brevemente estas dos separaciones.

2.4 Separación de intereses

2.4.1 Front-End

Es la parte del software que se enfoca en el usuario con lo que puede interactuar gráficamente. Existen diferentes frameworks, preprocesadores y librerías que ayudan a proporcionar experiencia de usuario, inmersión y usabilidad.

2.4.2 Back-End

Es la parte que se enfoca en procesar los datos provenientes de las entradas desde el front-end para que funcione correctamente el software. En términos generales es la parte donde están las

reglas de negocio y en esta parte interactúan lenguajes programación, así como frameworks y librerías que realizan este proceso.

Estas dos separaciones algunos autores lo mencionan como de lado del cliente (front-end) y de lado del servidor (back-end), más adelante se describirán algunas herramientas de desarrollo que actúan en estas dos separaciones.

Un dato relevante en la elaboración de aplicaciones es la arquitectura con la cual se trabajará, esta define una estructura de cómo debe ser construido el software, a continuación, se describen algunas de las arquitecturas que son utilizadas para construcción de software.

2.5 Arquitecturas de software

La arquitectura de software se apasiona por comprender cómo debe estructurarse una herramienta y cómo debe de organizarse la estructura general de una herramienta. En el proceso de modelado de desarrollo de sistemas, la arquitectura de software es la parte principal en el seguimiento de modelado del software. Sendero más importante en medio de la planeación y la ingeniería de requerimientos principales y secundarios, debido a que detecta los principales objetos arquitectónicos en una herramienta y la coexistencia entre estos.

La arquitectura de software es crucial debido a que interviene en la respuesta y potencia, así también la amplitud de asignación y mantenimiento de una herramienta (Bosch, 2000). Nos dice Bosch, los objetos propios ejercen los requerimientos funcionales de la aplicación. Los requerimientos no funcionales trabajan bajo el diseño de la herramienta (Sommerville, Ingenieria de Software, 2016).

Existen diferentes arquitecturas de software que proponen una manera diferente de organizar un sistema de software, las cuales se describirán en los siguientes apartados.

2.5.1 Arquitectura en capas

Organiza un sistema en capas donde las funcionalidades están relacionadas con cada capa. Una capa proporciona servicios a la capa superior, las capas de nivel más bajo representan funcionalidades o servicios centrales que pueden ser usados en todo el sistema, como se muestra en la figura 2.11 (Sommerville, Ingenieria de Software, 2016).

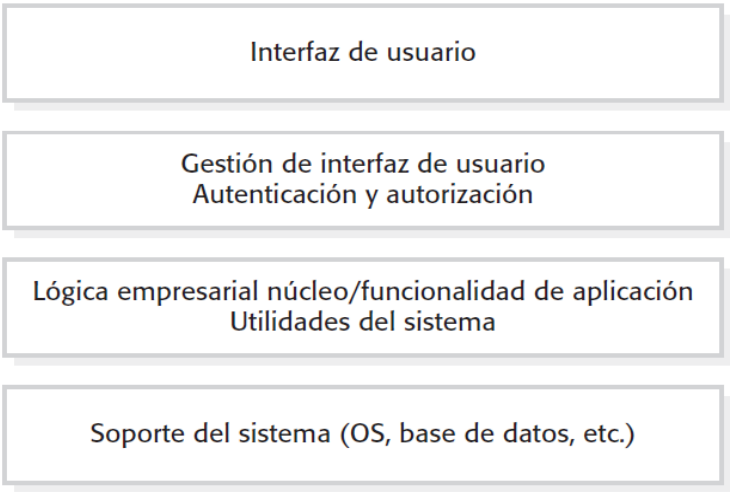

Figura 2.11 Arquitectura genérica en capas (Sommerville, 2016).

Un ejemplo para una estructura en partes de una herramienta para organizar archivos con privilegios de autor se tiene en diferentes bibliotecas, como se muestra en la figura 2.12.

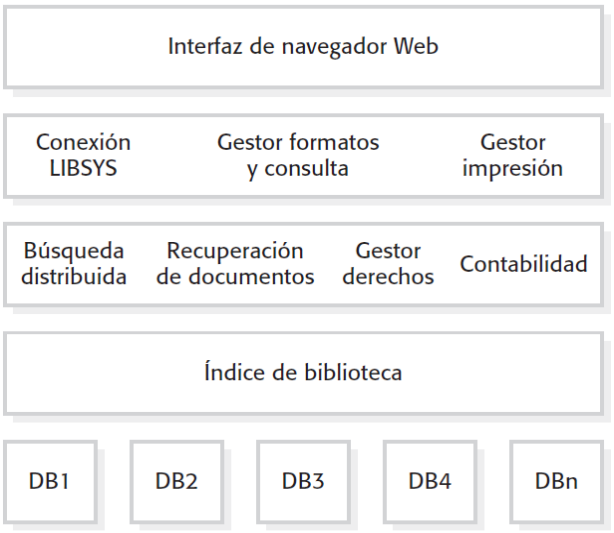

Figura 2.12 Arquitectura del sistema de biblioteca llamado LIBSYS (Sommerville, 2016).

2.5.2 Arquitectura cliente - servidor

En una arquitectura cliente-servidor, la función de una herramienta se estructura en diversos servicios, y cada servicio lo inserción a una maquina independiente. Los clientes son usuarios de dichos servicios y para implementarlos acceden a las máquinas. En la figura 2.13 se muestra de una filmoteca y videoteca (videos / DVD) estructurada como una aplicación cliente-servidor (Sommerville, Ingenieria de Software, 2016).

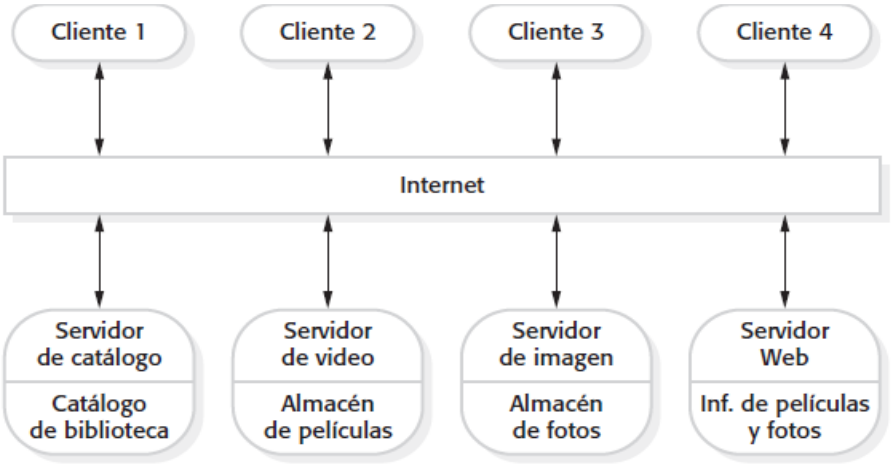

Figura 2.13 Arquitectura cliente-servidor para una filmoteca (Sommerville, 2016).

2.5.3 Arquitectura de tres capas

El patrón multicapa separa una herramienta mono capa en diversas capas. Su principal objetivo es distribuir los objetos en relación a su operación, digamos, en las herramientas existen objetos comisionados de la exposición, otros de representar la lógica de negocios y algunos de la persistencia de datos.

En situaciones se desordena el término “capas” con el término “nivel”. El primer término se utiliza para hacer referencia a las diferentes “partes” en las que una aplicación se divide desde el punto de vista lógico. Sin embargo, el segundo término es relacionado a la forma física en que una herramienta se estructura. Ejemplo sencillo (descrito mediante la figura 2.15) y que es

relativamente repetitivo es encontrar, es una herramienta que tiene dos niveles (nivel de aplicación y nivel de datos), en donde cada nivel puede tener varias capas.

Al separar una aplicación en capas y niveles se tiene la capacidad de modificar de forma independiente cada capa (Tahuiton Mora, 2011).

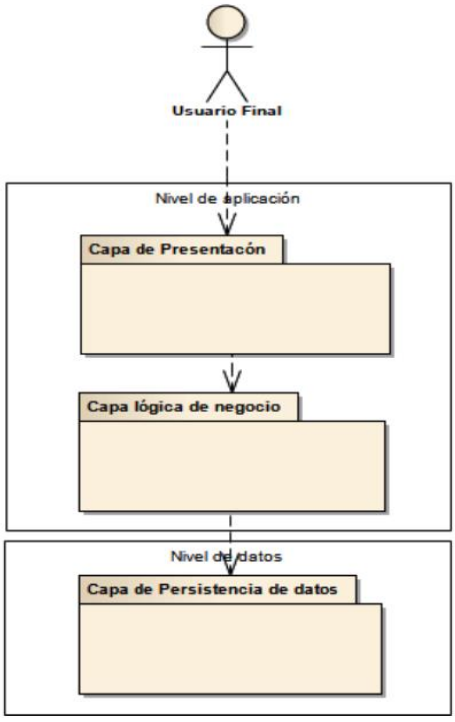

Figura 2.14 Arquitectura dividida en dos niveles y tres capas (Tahuiton Mora, 2011).

Para el desarrollo de la aplicación se utiliza el patrón Modelo-Vista-Controlador (MVC) ya que es adaptable para el desarrollo de aplicaciones web por tal motivo fue la arquitectura seleccionada ya que la aplicación que se pretende desarrollar está basada en web. Además, la mayoría de los frameworks existentes para la construcción de aplicaciones web de lado del front-end y back-end implementan este patrón arquitectónico ya que separa los intereses o conceptos dentro de una aplicación ayudando a que exista un orden en la codificación, sea fácil el mantenimiento y como gran parte de los patrones de diseño que sea reutilizable el código.

Patrón de diseño Modelo-Vista-Controlador (MVC)

Un patrón de diseño se caracteriza como "una regla de tres partes que expresa una relación entre cierto contexto, un problema y una solución" (Pressman, 2010). En pocas palabras, se considera un patrón de diseño como una solución estandarizada en el cual por medio de abstracción se logre crear código reutilizable.

El Modelo-Vista-Controlador o también conocido como MVC, es un patrón de diseño de software el cual hace una separación clara de los componentes de un sistema, de modo que cada uno de estos ejecute una clase de instrucciones y que al compilarse se logren unir en la ejecución. Esté separa los datos y la lógica del negocio de una aplicación de la interfaz del usuario y el módulo encargado de gestionar los eventos y las comunicaciones (Eslava, 2011). El patrón MVC se divide en tres capas que son importantes en una aplicación las cuales son:

- Modelo: Es el objeto que representa los datos del programa. Maneja los datos y controla todas sus transformaciones. Este no tiene conocimiento específico de los Controladores o de las Vistas, ni siquiera contiene referencias a ellos. Es el propio sistema el que tiene encomendada la responsabilidad de mantener enlaces entre el Modelo y sus Vistas, y notificar a las Vistas cuando cambia el Modelo.

- Vista: Es el objeto que maneja la presentación visual de los datos representados por el Modelo. Genera una representación visual del Modelo y muestra los datos al usuario. Interactúa preferentemente con el Controlador, pero es posible que trate directamente con el Modelo a través de una referencia al propio Modelo.

- Controlador: Es el objeto que proporciona significado a las órdenes del usuario, actuando sobre los datos representados por el Modelo, centra toda la interacción entre la Vista y el

Modelo. Cuando se realiza algún cambio, entra en acción, bien sea por cambios en la información del Modelo o por alteraciones de la Vista. Interactúa con el Modelo a través de una referencia al propio Modelo.

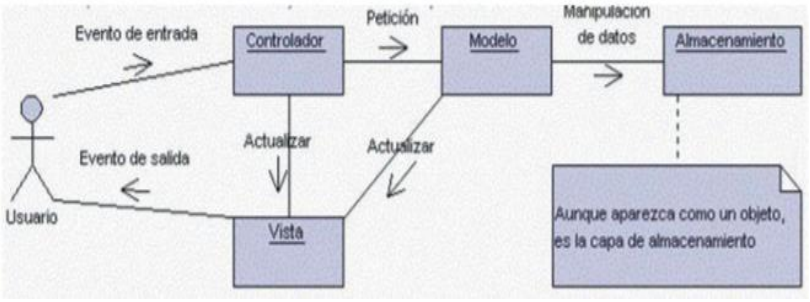

Figura 2.15 Interacción entre las capas de la arquitectura MVC (Fernández & Díaz, 2012).

Después de haber documentado las diferentes arquitecturas de software, en la siguiente sección se describirán los lenguajes de programación y frameworks que actúan en el front-end y posteriormente las que actúan en el back-end, con los cuales se realizará la elaboración de la herramienta propuesta en el objetivo de la tesis.

2.6 Tecnologías de desarrollo

Una aplicación web está compuesta por diferentes tecnologías, pero todas comparten una misma tecnología que es HTML y esto es debido a que el navegador sólo interpreta documentos en formato HTML ya que es lo que entiende.

2.6.1 Lenguaje de marcas de hipertexto o HTML (HyperText Markup Language)

Es un lenguaje utilizado para definir la presentación de información en páginas Web. Gracias a HTML se ha podido combinar texto y gráficos en una misma página y crear sistemas de presentación complejos con hiperenlaces entre páginas. Pero HTML no es útil en lo que se refiere a la descripción de información; XML sí. Por ejemplo, se puede utilizar HTML para dar formato

a una tabla, pero no para describir los elementos de datos que componen la misma (Ceballos, 2007).

Para dar una mayor presentación a las páginas Web que forman a una aplicación basada en Web se necesita establecer estilos utilizando Hojas de estilo en cascada (CSS) debido a que en ocasiones los estilos predeterminados en las etiquetas que proporciona HTML no son suficientes.

2.6.2 Hojas de estilo en cascada o CSS (Cascading Style Sheets)

Utilizando CSS, se puede aplicar estilos a las páginas web para dar estilos específicos. Esto funciona porque CSS está conectado al **Modelo de Objetos de Documento** (DOM), con CSS se puede restaurar cualquier elemento rápidamente y fácilmente. Por ejemplo, si se desea tener el aspecto predeterminado de una etiqueta HTML con CSS se puede asignar nuevos estilos para anular la configuración predeterminada. Una forma de agregar estilo a una página web es insertando las declaraciones necesarias usando las etiquetas <style> y </style> en el encabezado de la página, entre las etiquetas <head> y </head>. Con la aparición del estándar CSS3 en los últimos años, CSS ahora ofrece un nivel de interactividad dinámica que antes sólo era compatible con JavaScript. Por ejemplo, no sólo puede aplicar un estilo a cualquier elemento HTML para cambiar sus dimensiones, colores, bordes, espaciado, etc. Ahora también puede agregar transiciones y transformaciones animadas a sus páginas web, utilizando sólo unas pocas líneas de CSS. (Nixon, 2012)

En ocasiones no basta con sólo dar una estructura y un estilo a las páginas web que componen una aplicación, ya que existen formularios para entrada de datos y estos datos tiene que ser validados o procesados antes de ser enviados al servidor y JavaScript permite realizar este tipo de acción.

2.6.3 JavaScript

JavaScript es un lenguaje de script del lado del cliente que se ejecuta completamente dentro del navegador web. Para llamarlo, se tiene que colocar el código JavaScript al abrir las etiquetas HTML <script> y cerrar </script>.

JavaScript fue creado para permitir el acceso a todos los elementos de un documento HTML. En otras palabras, proporciona un medio para la interacción dinámica del usuario, como verificar la validez de la dirección de correo electrónico en los formularios de entrada. Combinado con CSS, JavaScript es el poder detrás de las páginas web dinámicas (Nixon, 2012).

Cuando se desarrolla una aplicación web en ocasiones es necesario almacenar información en el navegador para poder realizar operaciones o validaciones en el Front-End, con el uso del almacenamiento web o web Storage es posible hacer lo mencionado.

2.6.4 Almacenamiento web o web Storage

Con el almacenamiento web, las aplicaciones web pueden almacenar datos localmente dentro del navegador del usuario. Antes de HTML5, los datos de la aplicación debían almacenarse en cookies, incluidas en cada solicitud del servidor. El almacenamiento web es más seguro y se pueden almacenar grandes cantidades de datos localmente, sin afectar el rendimiento del sitio web.

A diferencia de las cookies, el límite de almacenamiento es mucho mayor (al menos 5 MB) y la información nunca se transfiere al servidor. El almacenamiento web es por origen (por dominio y protocolo). Todas las páginas, de un origen, pueden almacenar y acceder a los mismos datos. El almacenamiento web HTML proporciona dos objetos para almacenar datos en el cliente:

- Window.localStorage: Almacena datos sin fecha de vencimiento.

- Window.sessionStorage: Almacena datos para una sesión (los datos se pierden cuando se cierra la pestaña del navegador) (Refsnes Data, 2020).

Con CSS se mencionó con anterioridad que proporciona diferentes estilos, pero crear estilos en ocasiones puede ser muy tardado, entonces para poder dar estilos a de una manera más fácil y rápida, actualmente existen diferentes frameworks que dan estilos a las páginas web con facilidad y rapidez, entre los frameworks existentes está Vuetify. A continuación, se describe este framework.

2.6.5 Vuetify

Vuetify es un framework que combina **la potencia del popular VueJs con la estética de Material Design**. Permite acelerar el desarrollo de aplicaciones web complejas, incorporando una gran cantidad de componentes "listos para usar".

Vuetify se basa en el habitual sistema tipo "grid" para la ordenación del layout de la página. Dispone de una **enorme librería de componentes** que incluye desde elementos de formulario sencillos como botones, combobox, inputs, sliders, a componentes más avanzados típicos en aplicaciones Android como "cards" o "snackbars".

En conclusión, Vuetify es **uno de los framework más potentes** que hay ahora mismo para desarrollo web. Combinado con VueJS CLI UI tenemos todos los medios para empezar a programar aplicaciones web realmente potentes de forma sencilla.

2.6.6 JQuery

JQuery es una biblioteca de JavaScript que contiene funciones que permiten la manipulación del DOM de los documentos HTML, el manejo de eventos, animación y AJAX sean mucho más

simples de usar mediante la implementación de una API permitiendo que funcione en diferentes navegadores. Con la combinación de versatilidad y extensibilidad puede ser utilizada por cualquier lenguaje de programación que actué en el back-end (Resig, 2006).

En ocasiones es necesario enviar información de los formularios al servidor sin necesidad de refrescar la página Web para que no cargue por completo todos los elementos que compone una página Web donde JavaScript asíncrono y XML (AJAX) resuelve este problema de envío de información.

2.6.7 JavaScript asíncrono y XML o Ajax (Asynchronous JavaScript And XML)

Esta tecnología se ejecuta de lado del cliente, es decir, en el navegador los usuarios pueden mantener la comunicación asíncrona con el servidor en segundo plano, en el sentido de que los datos adicionales se solicitan al servidor y se cargan en segundo plano sin interferir con la visualización ni el comportamiento de la página (Galeano Arenas, 2016). AJAX permite enviar y recibir información mediante los métodos POST y GET del protocolo HTTP.

En ocasiones es necesario mostrar resultados sencillos y de fácil lectura en las aplicaciones, para ello, se utilizan gráficas que representen información de manera agradable y entendible. Hoy en día existen diferentes librerías que permiten crear diferentes tipos de gráficas para la visualización de la información en ellas, axios es una de estas librerías que permite lo mencionado anteriormente.

2.6.8 Axios

Axios es un cliente HTTP basado en Promesas para Javascript, el cual puede ser utilizado tanto en tu aplicación Front-end, como en el Back-end con Nodejs. Utilizando Axios, es muy sencillo enviar peticiones a endpoints REST y realizar operaciones CRUD. Además, Axios puede ser

utilizada desde una aplicación desarrollada con Javascript plano, al igual que utilizando un Framework como Vuejs. (Axios, 2017)

Características principales de Axios:

- Realizar peticiones XMLHttpRequest (Ajax) desde el navegador de una manera sencilla.
- Realizar peticiones HTTP desde Nodejs.
- Soporta el API de Promesas.
- Intercepta peticiones y respuestas.
- Transforma la información de las peticiones y respuestas.
- Cancela peticiones.
- Transforma automáticamente la información en formato JSON.
- Soporta protección del lado del cliente contra ataques CSRF (Cross-site request forgery).

2.6.9 Node.js

Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con E/S de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como, por ejemplo, servidores web y ejecutándose en el mismo. (Node.js, 2020)

2.6.10 Vue.js

Vue (pronunciado /vjuː/, como view) es un framework progresivo para construir interfaces de usuario. A diferencia de otros frameworks monolíticos, Vue está diseñado desde cero para ser utilizado incrementalmente. La librería central está enfocada solo en la capa de visualización, y es fácil de utilizar e integrar con otras librerías o proyectos existentes. Por otro lado, Vue también es

perfectamente capaz de impulsar sofisticadas Single-Page Applications cuando se utiliza en combinación con herramientas modernas y librerías de apoyo. (Vue.js, 2020)

Vue.js es utilizado para el desarrollo para el front-end de este proyecto, debido a que será un SPA (Single-Page Application).

2.6.11 Lenguaje C#

Como dice (Ferguson, Patterson, Beres, Boutquin, & Gupta, 2003) los pasados veinte años, C++ ha sido uno de los lenguajes elegidos para desarrollar herramientas comerciales y de negocios.

La comunidad de desarrolladores requería de un lenguaje de programación que fuese un intermedio entre C y C++. Un lenguaje que ayuda a desarrollar herramientas rápidas pero que a su vez ayudase a dar un gran control y un lenguaje que se incorporara bien con la producción de herramientas Web, XML y muchos otros lenguajes emergentes.

La razón por la cual elegí C# es porque es uno de los lenguajes con mayor uso en la actualidad, y uno de los que más información y librerías existe para el desarrollo de software.

2.6.12 ASP.NET Core MVC

Como dice (Lock, 2018) el desarrollo de ASP.NET Core fue motivado por el deseo de crear un marco web con cuatro objetivos principales:

- Para ser ejecutado y desarrollado multiplataforma.
- Tener una arquitectura modular para facilitar el mantenimiento.
- Para ser desarrollado completamente como software de código abierto.
- Para ser aplicables a las tendencias actuales en el desarrollo web, como las aplicaciones del lado del cliente y la implementación en entornos de nube.

Para lograr todos estos objetivos, **Microsoft** necesitaba una plataforma que pudiera proporcionar bibliotecas subyacentes para crear objetos básicos como listas y diccionarios, y realizar, por ejemplo, operaciones de archivos simples. Hasta este punto, el desarrollo de ASP.NET siempre se había centrado y dependía de *.NET Framework* solo para *Windows*. Para ASP.NET Core, Microsoft creó una plataforma ligera que se ejecuta en *Windows*, *Linux* y *macOS* llamada .NET Core, como se muestra en la Figura 2.16 *.NET Core* comparte muchas de las mismas API que .NET Framework, pero es más pequeño y actualmente solo implementa un subconjunto de las características que proporciona .NET Framework, con el objetivo de proporcionar una implementación y un modelo de programación más simples. Es una plataforma completamente nueva, en lugar de una bifurcación de .NET Framework, aunque utiliza un código similar para muchas de sus API.

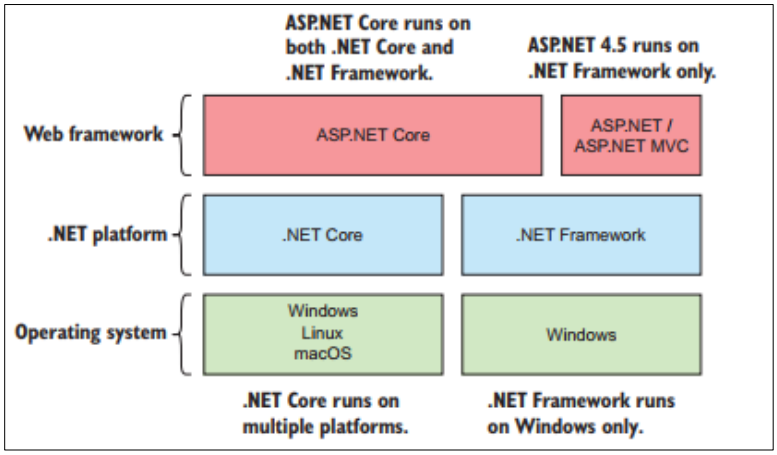

Figura 2.16 Arquitectura de ASP.NET Core

2.6.13 Entity Framework Core

Entity Framework Core, también conocido como EF Core, es un paquete de ORM producido por Microsoft que permite que las aplicaciones .NET Core almacenen datos en bases de datos relacionales. Entity Framework Core tiene una tarea clave: almacenar objetos .NET en una base

de datos y recuperarlos más tarde. Dicho de otra manera, EF Core actúa como el puente entre una aplicación MVC de ASP.NET Core y una base de datos, como se muestra en la Figura 2.17.

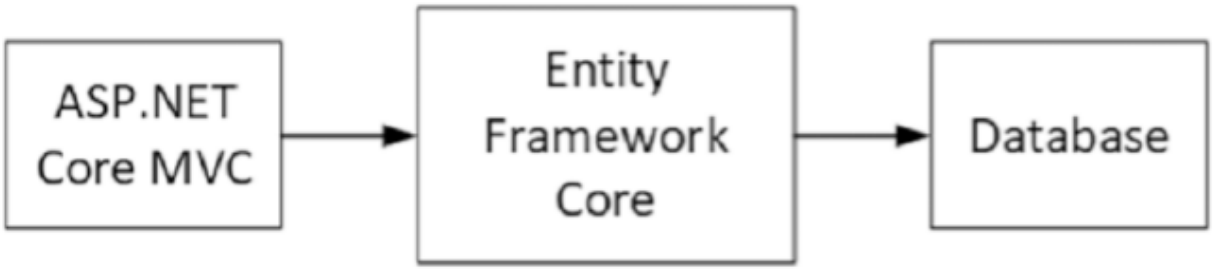

Figura 2.17 Entity Framework Core en contexto.

Entity Framework Core proporciona acceso fácil a la base de datos usando un lenguaje de **Consulta de Idioma Integrado** (en inglés, Languaje Integrated Query o LINQ) que es escrito similarmente a SQL. EF Core convierte el LINQ a SQL y se ejecuta en la base de datos. Cuando el SQL es ejecutado, EF Core toma la respuesta de la consulta y convierte los resultados dentro de un modelo para facilitar el acceso dentro del código (Freeman, 2018).

EF Core proporciona tres diferentes flujos de trabajo donde se puede configurar y usar dentro de un proyecto:

Database First: Este flujo es para cuando se tiene una base de datos existente o se desea tener un control total sobre como la base de datos es creada y mantenida. Cuando se usa este flujo, se puede crear un archivo EDMX que almacena el esquema de datos, modelo de datos y la relación entre el esquema y los modelos en XML.

Model First: Este flujo es un poco similar al Database First en que los modelos y las relaciones son mantenidas dentro de un archivo EDMX. En lugar de generar EDMX automáticamente de un diseño de la base de datos, se crean manualmente los modelos y se define un inter-modelo de relaciones usando el diseñador de Visual Studio. Una vez finalizado, se tiene que indicar a EF Core que cree las tablas, columnas, llaves primarias y foráneas necesarias en la base de datos.

Code First: Con Code First se puede indicar a EF Core que cree automáticamente la base de datos. O si se tiene una base de datos existente esta se puede modificar, se pueden usar las herramientas de EF Core para crear las clases iniciales de Code First. Cuando se usa Code First, Entity Framework proporciona un conjunto de herramientas que permiten realizar las Migraciones para modificar o actualizar automáticamente la base de datos cuando el modelo cambia (Munro, 2015).

Para facilitar la gestión y manipulación de datos para el proyecto se usará EF Core además usando el enfoque Code First debido a que da mayor control sobre las clases y objetos modelados para base de datos y proporciona una mayor facilidad para el control de las versiones de la base de datos de tal manera que si no se desea trabajar con una versión de la base de datos se puede regresar a versiones anteriores.

2.6.14 Microsoft Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, Integrated Development Environment por sus siglas en inglés) para sistemas operativos Windows. Un completo IDE extensible y gratuito con todas las características para crear aplicaciones modernas para Windows, Android e iOS, además de aplicaciones web y servicios en la nube.

La razón principal por la cual utilizare Microsoft Visual Studio es por las ventajas que nos proporciona a la hora de desarrollar una aplicación, nos permite desarrollar aplicaciones de escritorio y aplicaciones web sin problemas, contando con una amplia fuente de información actualmente. (Microsoft Visual Studio, 2019)

2.6.15 Microsoft SQL Server

Microsoft SQL Server es un sistema de manejo de bases de datos del modelo relacional, desarrollado por la empresa Microsoft. El lenguaje de desarrollo utilizado por la línea de comandos o mediante la interfaz gráfica de "*Management Studio*" es Transact-SQL (TSQL). Dentro de los competidores más destacados de SQL Server están: Oracle, MariaDB, MySQL, PostgreSQL. SQL Server ha estado tradicionalmente disponible solo para sistemas operativos Windows de Microsoft, pero desde 2017 también está disponible para Linux y Docker containers. Puede ser configurado para utilizar varias instancias en el mismo servidor físico, la primera instalación lleva generalmente el nombre del servidor.

La razón por la cual utilizare este motor de base de datos es porque alimentaré mi sistema y base de datos con la que actualmente se trabaja en la organización donde se establecerá el sistema, cuestión de compatibilidad.

2.6.15.1 Características

Este sistema incluye una versión reducida, llamada Motor de escritorio de Microsoft (MSDE) con el mismo motor de base de datos, pero orientado a proyectos más pequeños, que en sus versiones 2005 y 2008 pasa a ser el SQL Express Edition, que se distribuye en forma gratuita. Es común desarrollar proyectos completos empleando Microsoft SQL Server y Microsoft Access a través de los llamados Proyecto de acceso a datos (ADP) (Microsoft SQL Server, 2018).

2.7 Enterprise Architect

Sparx Systems Enterprise Architect es una herramienta de diseño y modelado visual basada en OMG UML. La plataforma soporta: el diseño y construcción de sistemas de software; modelado de procesos de negocio; y modelado de dominios basados en la industria.

La razón por la cual voy a usar esta herramienta es para crear un mejor diseño de mis diagramas UML y casos de uso de mi proyecto (Enterprise Architect, 2019).

Capítulo 3 Análisis y Diseño

La documentación generada en esta etapa corresponde a la propuesta por el proceso de desarrollo de Rational, cabe mencionar que RUP sugiere ciertos modelos y diagramas, aunque dependiendo del tipo de aplicación y de las necesidades de la empresa a la que se le está desarrollando, de tal forma que no necesariamente se tienen que elaborar todos y cada uno de los modelos y artefactos de RUP.

3.1 Modelado de Negocios

Con estos modelos se muestra una vista general del proceso de contratación de un servicio VPS y el proceso de renta de la empresa, en éste se visualiza en dónde el sistema **SYS-VPS** ayudará a automatizar ciertas operaciones del área administrativa de BTU Comunicación.

Este diagrama muestra los procesos en los que se divide la operación del área administrativa, en la empresa BTU Comunicación, (ver figura 3.1). Solo en los procesos que se muestran en azul son donde SYS-VPS intervendrá en las operaciones que ocurren en el área administrativa.

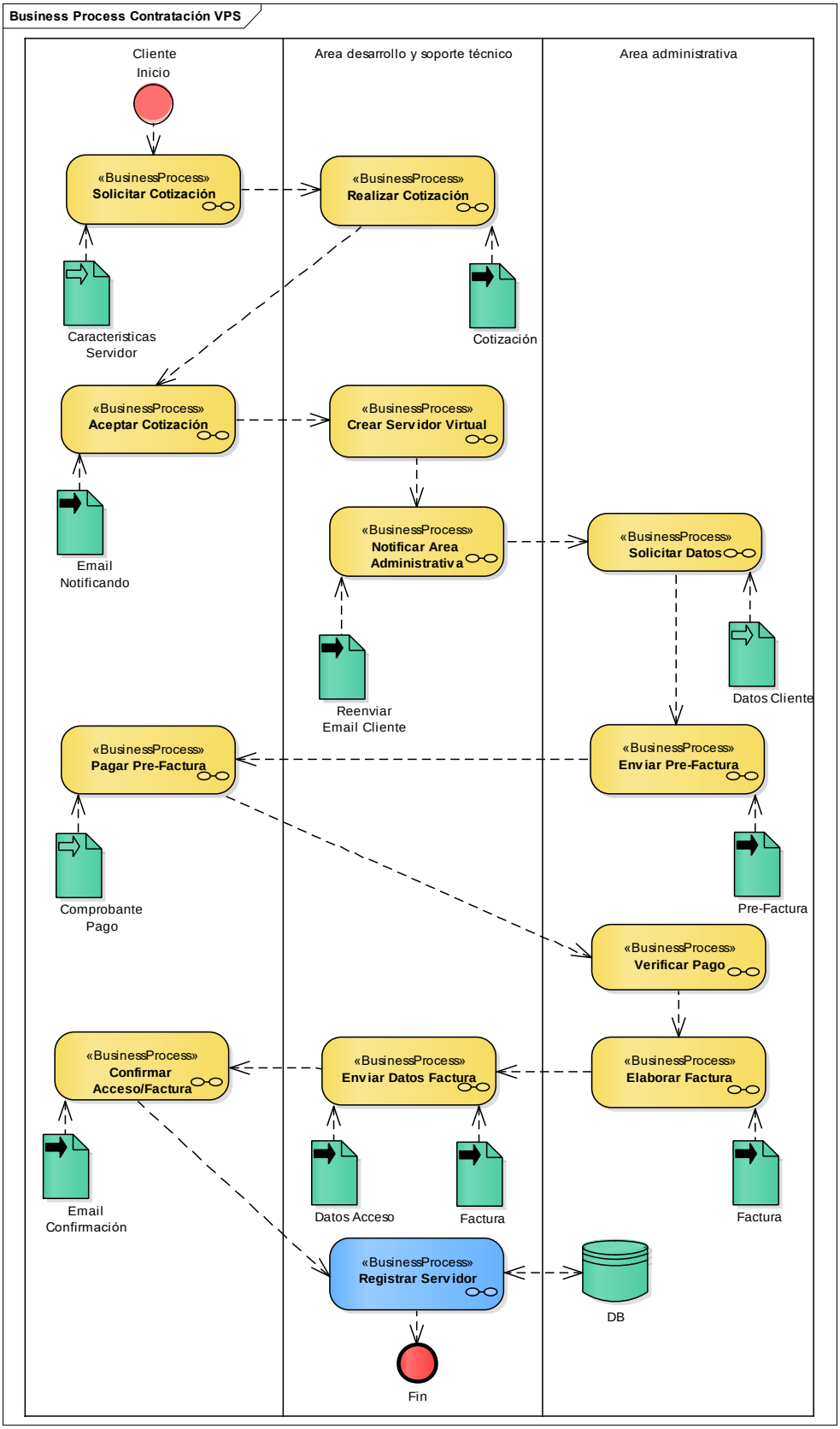

Figura 3.1 Diagrama de modelado de negocio - Contratación VPS.

A continuación se explica brevemente en qué consiste cada proceso del modelado de negocio.

- **Solicitar Cotización**

  El cliente se comunica a la Empresa BTU Comunicación para solicitar una cotización, el cliente es atendido por el Área de Desarrollo y Soporte Técnico.

- **Realizar Cotización**

  El personal del Área de Desarrollo y Soporte Técnico se encarga de realizar la cotización en base a las características proporcionadas por parte del cliente.

- **Aceptar Cotización**

  El cliente acepta la cotización y solicita se comience el desarrollo del Servidor Virtual Privado.

- **Crear Servidor Virtual**

  El personal del área de desarrollo y soporte comienza el desarrollo del Servidor Virtual Privado.

- **Notificar Área Administrativa**

  El personal del área de desarrollo notifica al encargado del área administrativa que el cliente ha aceptado la cotización y puede comenzar a desarrollar la factura correspondiente.

- **Solicitar Datos**

  El área administrativa se comunica con el cliente vía correo electrónico o vía telefónica para solicitar los datos necesarios para la realización de la factura y de forma opcional a consideración del cliente, el contrato.

- **Pagar Pre - Factura**

  El Cliente paga la pre - factura y notifica al área administrativa.

- **Enviar Pre-Factura**

  El área administrativa crea la pre - factura y envía a su respectivo cliente. Solicitando se le notifique cuando se realice el depósito.

- **Verificar Pago**

  El área de desarrollo verifica que se ha realizado el pago de la factura y notifica al área de desarrollo y soporte que puede entregar los datos de acceso al servidor virtual privado al cliente.

- **Elaborar Factura**

  El área administrativa desarrolla la factura una vez que se ha confirmado el pago.

- **Enviar Datos/Factura**

  El Área de Desarrollo al ser notificado se pone en contacto con el cliente por medio de correo electrónico y proporciona los datos de acceso y factura al cliente, así mismo se solicita que el cliente verifique si puede acceder sin problemas al servidor y su factura esté correcta.

- **Confirmar Acceso/Factura**

  El cliente le confirma al Área de Desarrollo que puede entrar sin problemas al Servidor Virtual Privado y que su factura es correcta.

- **Registrar Servidor**

  El departamento de desarrollo y soporte técnico registra los datos del Servidor Virtual Privado.

El proceso de Renta de un VPS se muestra en el siguiente diagrama de modelado, (ver figura 3.2). Donde se podrá apreciar de color azul los procesos en los cuales interviene **SYS-VPS**.

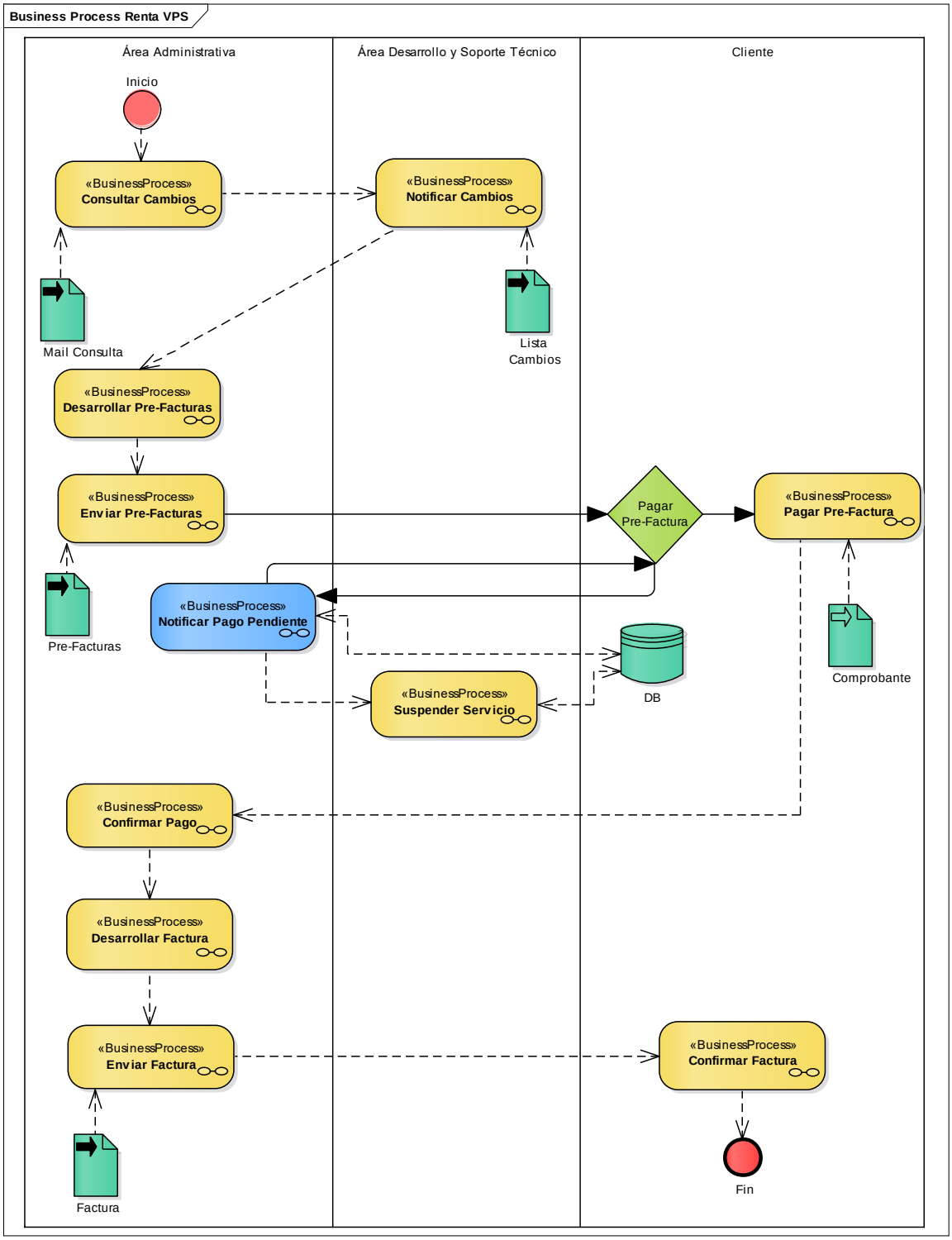

Figura 3.2 Diagrama de modelado de negocio - Renta VPS.

A continuación se explica brevemente en qué consiste cada proceso del modelado de negocio.

- **Consultar Cambios**

  El Área Administrativa consulta al área de desarrollo si algún cliente ha requerido cambios en las especificaciones de su servidor virtual privado.

- **Notificar Cambios**

  El Área de desarrollo y soporte técnico notifica al Área Administrativa si existen cambios o no en los servidores virtuales para realizar los ajustes correspondientes a las facturas antes de realizarla.

- **Desarrollar Pre - Facturas**

  El Área Administrativa desarrolla las pre - facturas tomando en consideración si se realizaron cambios durante el periodo.

- **Enviar Pre-Facturas**

  El Área Administrativa se encarga de distribuir las pre - facturas correspondientes al mes de servicio de servidor virtual privado.

- **Notificar Pago Pendiente**

  El área administrativa notifica al cliente que no ha recibido el pago correspondiente a su servicio VPS.

- **Suspender Servicio**

  El área administrativa notifica al área de soporte técnico que determinado cliente no ha realizado su pago correspondiente, y le pide suspender su servicio.

- **Pagar Pre-Factura**

  El Cliente recibe la pre - factura y realiza el pago del servicio. De la misma forma notifica al Área Administrativa que ha realizado el pago.

- **Confirmar Pago**

  El Área Administrativa confirma que se ha realizado el depósito correspondiente al servicio de forma correcta.

- **Desarrollar Factura**

  Tras confirmar el pago por parte del cliente, el área administrativa genera la factura del cliente.

- **Enviar Factura**

  El área administrativa envía la factura al cliente.

- **Confirmar Factura**

  El cliente recibe la factura de su servicio.

3.2 Especificación de requerimientos

Los requisitos para desarrollar la aplicación SYS-VPS están alineados al objetivo general y a los objetivos específicos, el general es:

Desarrollar una herramienta que notifique de forma automática vía correo electrónico a los clientes que estén por expirar su periodo de pago por el servicio de servidor virtual privado en BTU Comunicación.

Los objetivos específicos son:

- Notificar a los clientes mediante correo electrónico que su pago se encuentra pendiente.

  Una vez proporcionada la pre - factura al cliente contará con un total de quince días para liberar su pago, se les notificará faltando cinco y dos días antes de terminar el periodo.
- Registrar, Modificar y Eliminar clientes y servidores virtuales privados alojados en la organización.

  Cuando la organización rente un nuevo servidor y se establezca como tipo renta, el sistema lo tomará en cuenta para notificar a su respectivo propietario.
- Llevar un control de las notificaciones relacionadas con los Servidores.

  Las notificaciones vía correo electrónico serán almacenadas en una cuenta de correo propia de la organización y funcionarán como evidencia en caso de alguna incidencia con algún cliente.

3.3 Análisis

3.3.1 Modelo de casos de uso

Para el modelado de casos de uso se elaboró uno por cada proceso de negocio, la figura 3.3 muestra los casos de uso del módulo *Servidores Virtuales Privados.*

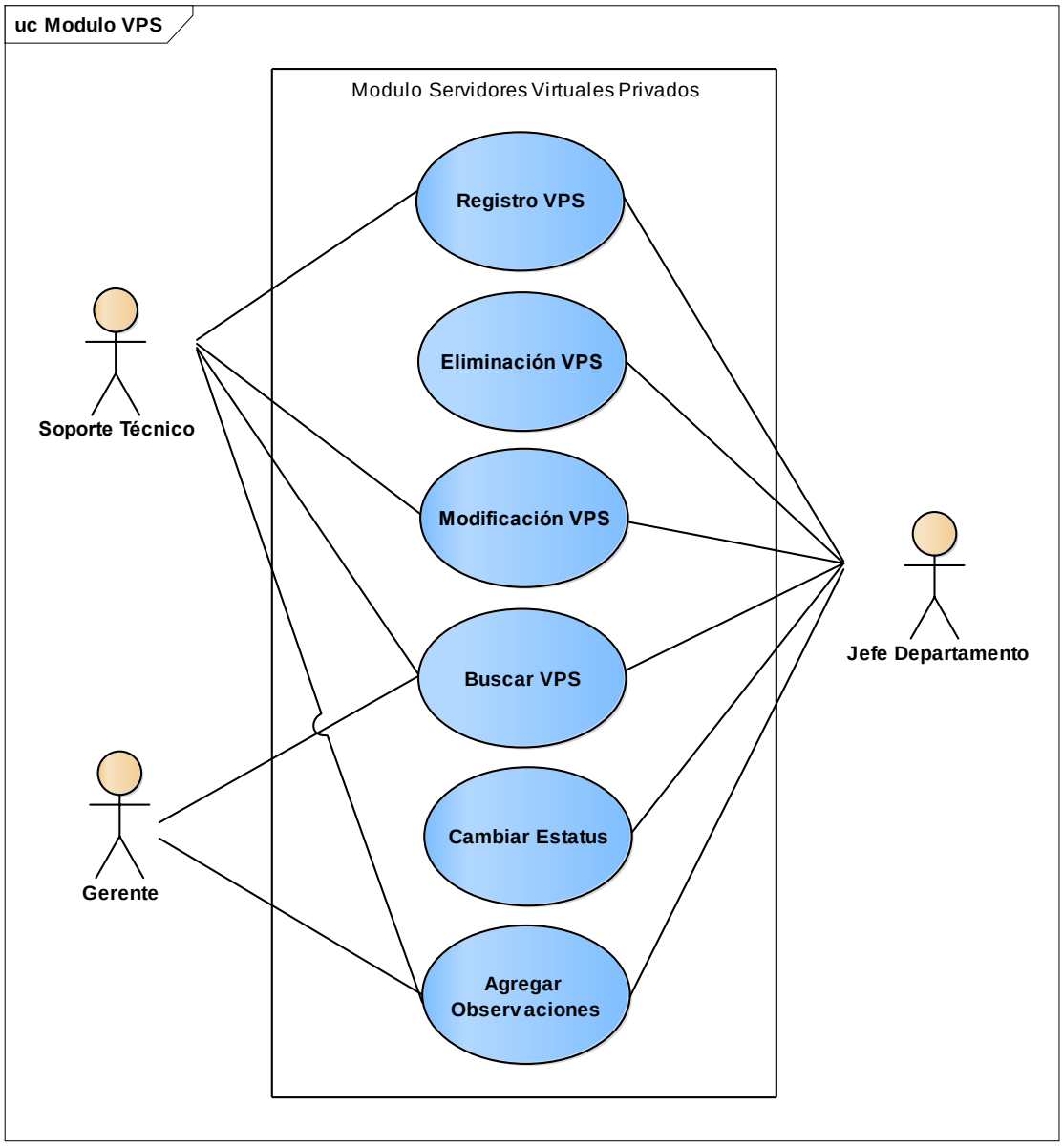

Figura 3.3 Modelado de los casos de uso: modulo Servidores Virtuales Privados.

La descripción de los actores es la siguiente:

- Jefe Departamento

  Persona calificada para realizar cualquier acción dentro de la aplicación, director de operaciones de los departamentos incluido el área de sistemas.

- Gerente

  Persona encargada del área de sistemas y encargada del personal del área de soporte técnico.

- Soporte Técnico

  Personal que supervisa y registra las operaciones de los servidores virtuales.

Descripción de caso de uso, según la forma de Craig Larman:

- **Registrar contrato**

  Paso 01: Dentro del módulo VPS da clic en el botón "Crear nuevo".

  Paso 02: Se registra automáticamente el ID del VPS dentro de la lógica del formulario.

  Paso 03: Se registra el nombre del VPS, nombre que nos ayudará a localizar el Servidor Virtual.

  Paso 04: Se registra el sistema operativo del servidor.

  Paso 05: Se registra el número de CPU del servidor.

  Paso 06: Se registra el número de GB de memoria RAM del servidor.

  Paso 07: Se registra el número de GB de disco duro del servidor.

  Paso 08: Se registra la cantidad de MB de ancho de banda.

  Paso 09: Se registra la cantidad de licencias de RD adquiridas por el cliente.

  Paso 10: se registra la IP Privada y la IP pública.

  Paso 11: Da clic en el botón guardar para almacenar este registro.

- **Eliminación VPS**

  Paso 01: Dentro del módulo servidores selecciona el registro del cliente.

  Paso 02: Da clic en el botón eliminar.

  Paso 03: Al dar clic la aplicación te llevará a otro formulario donde aparecerá solo el registro que desees eliminar.

  Paso 04. Da clic nuevamente en el botón eliminar para confirmar nuevamente que deseas eliminar este registro.

- **Modificación VPS**

  Paso 01: Dentro del módulo servidores selecciona un registro.

  Paso 02: Da clic en el botón editar.

  Paso 03: Al dar clic la aplicación te llevará a otro formulario donde aparecerá solo el registro que desees modificar.

  Paso 04. Dentro de este formulario podrás modificar cualquier dato a excepción del ID del registro.

  Paso05: Una vez modificado los datos da clic en actualizar y los datos habrán sido guardados.

- **Buscar contrato**

  Paso 01: Escribe el nombre del servidor virtual o la razón social de la VPS que desees buscar.

  Paso 02: Dentro del formulario VPS da clic en el botón "buscar".

- **Cambiar Estatus**

  Paso 01: Seleccionar el registro al que cambiaremos el status.

  Paso 02: Elegir el nuevo Estatus que deseemos.

Paso 03: Damos clic en guardar para cambiar el estatus del servidor.

- **Agregar observaciones**

  Paso 01: Seleccionaremos el registro al cual añadiremos la observación.

  Paso 02: Daremos clic en agregar observación

  Paso 03: Una vez redactada la observación daremos clic en guardar.

Módulo *Clientes* (figura 3.4).

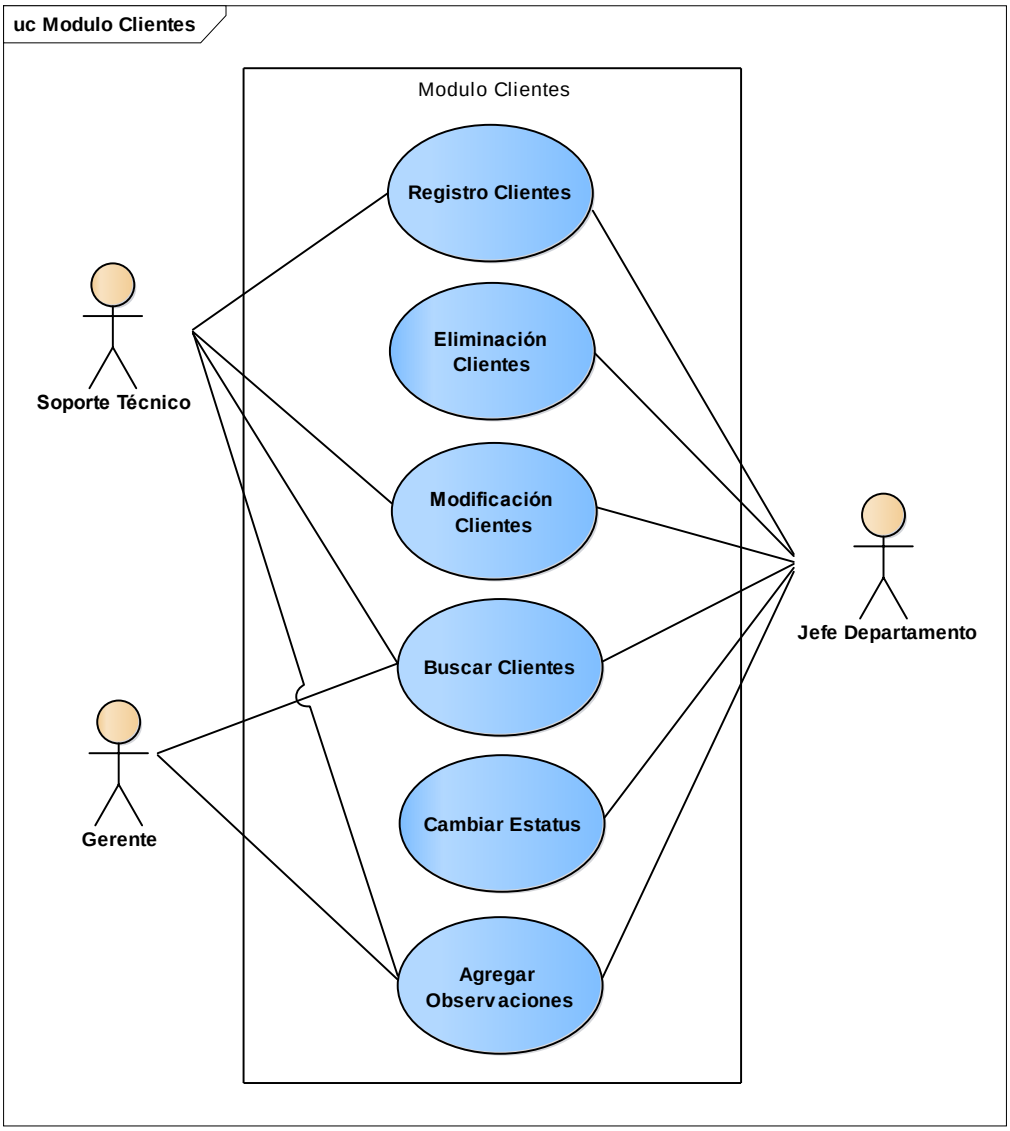

Figura 3.4 Módulo: Clientes.

Descripción de caso de uso, según la forma de Craig Larman:

- **Registrar cliente**

  Paso 01: Dentro del módulo VMClient da clic en el botón "Crear nuevo".

  Paso 02: Se registra automáticamente el ID del Cliente dentro de la lógica del formulario.

Paso 03: Se registra el nombre del Cliente, nombre que nos ayudará a localizar el Propietario del servicio.

Paso 04: Se registra el nombre completo de la organización.

Paso 05: Se registra correo electrónico de la organización.

Paso 06: Se registra el número de teléfono de la organización.

Paso 07: Se registra el nombre del contacto técnico de la organización.

Paso 08: Se registra el correo electrónico del contacto técnico de la organización

Paso 09: Da clic en el botón guardar para almacenar este registro.

- **Eliminación VPS**

  Paso 01: Dentro del módulo VMClient selecciona el registro del cliente.

  Paso 02: Da clic en el botón eliminar.

  Paso 03: Al dar clic la aplicación te llevará a otro formulario donde aparecerá solo el registro que desees eliminar.

  Paso 04. Da clic nuevamente en el botón eliminar para confirmar nuevamente que deseas eliminar este registro.

- **Modificación VPS**

  Paso 01: Dentro del módulo VMClient selecciona un registro.

  Paso 02: Da clic en el botón editar.

  Paso 03: Al dar clic la aplicación te llevará a otro formulario donde aparecerá solo el registro que desees modificar.

  Paso 04. Dentro de este formulario podrás modificar cualquier dato a excepción del ID del registro.

Paso05: Una vez modificado los datos da clic en actualizar y los datos habrán sido guardados.

- **Buscar contrato**

  Paso 01: Escribe el nombre del cliente o contacto técnico de la organización que desees buscar.

  Paso 02: Dentro del formulario VPS da clic en el botón "buscar".

- **Cambiar Estatus**

  Paso 01: Seleccionar el registro al que cambiaremos el status.

  Paso 02: Elegir el nuevo Estatus que deseemos.

  Paso 03: Damos clic en guardar para cambiar el estatus del servidor.

- **Agregar observaciones**

  Paso 01: Seleccionaremos el registro al cual añadiremos la observación.

  Paso 02: Daremos clic en agregar observación

  Paso 03: Una vez redactada la observación daremos clic en guardar.

3.3.2 Requerimientos de Software

Los requerimientos para poder desarrollar y utilizar la aplicación son los siguientes:

- PC de desarrollo
    - S. O. Windows 10
    - Enterprise architect 12.0.1210
    - Visual Studio Community 2019 versión 16.2.5
- Servidor de aplicaciones
    - Windows Server 2012 R2
    - Internet Information Services
- Servidor de base de datos
    - SQL Server Express Edition 2014

3.3.3 Requerimientos de Hardware

Los requerimientos para poder implantar la aplicación son los siguientes:

- 1 Servidor de aplicaciones (Virtual Server)
    - Procesadores
    - Xeon Procesador
    - S.O. Microsoft Windows Server 2012 R2
    - 8 GB de RAM
- 1 Servidor de base de datos (Virtual Server)
    - 4 procesadores
    - Xeon Procesador
    - S.O. Microsoft Windows Server 2012 R2
    - 16 GB de RAM

3.4 Diseño del sistema

Dentro del presente apartado se explicará la arquitectura utilizada que soporta a **SYS-VPS**.

3.4.1 Diagrama de clases

Se elaboró únicamente sólo diagrama de clases que contempla el proceso analizado: generación de contratos de servidores virtuales. Se colocaron en las clases sólo los atributos y funciones que tienen una relevancia importante para el modelo, esto es para hacerlas más legibles.

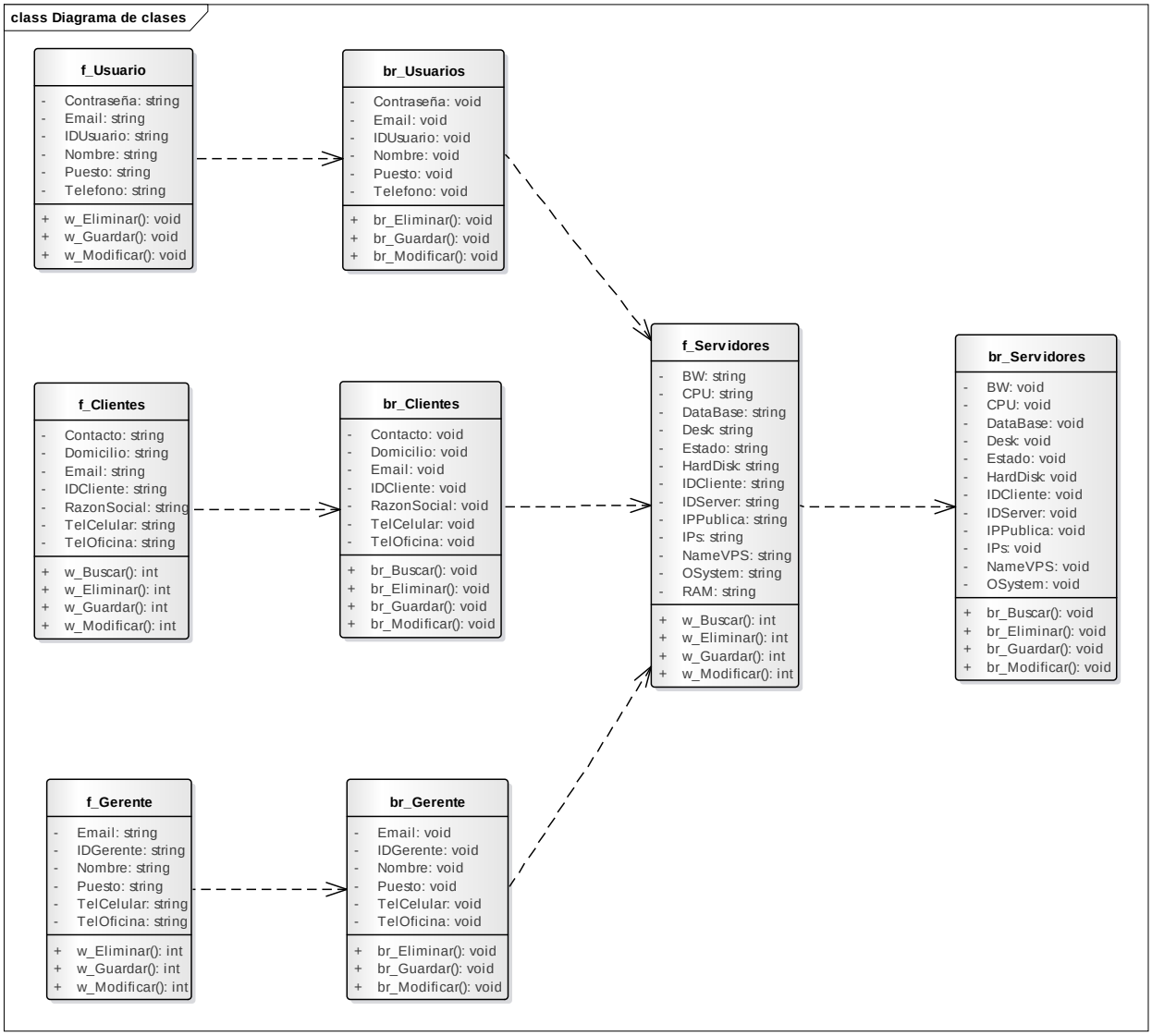

Figura 3.5 Diagrama de clases.

Descripción de las clases de los módulos:

- **f_Usuario**

  Esta es una clase tipo formulario, donde se tiene que capturar el correo electrónico y contraseña del usuario que desea tener acceso a la herramienta en un futuro, el formulario manda a validad los datos capturados por medio del componente br_Usuarios.

- **br_Usuarios**

  Es un componente que se encarga de manejar todas las reglas de negocio del módulo usuario, para permitir alojar la información de manera correcta en el arreglo de datos.

- **f_Clientes**

  Esta es una clase tipo formulario, donde se tiene que capturar la razón social, ubicación, contacto responsable de la empresa, correo electrónico, teléfonos y el estado actual, el formulario manda a validad los datos capturados por medio del componente br_Clientes.

- **br_Clientes**

  Es un componente que se encarga de manejar todas las reglas de negocio del módulo clientes, para permitir alojar la información de manera correcta en el arreglo de datos y poder asignarle algún servicio a su cargo.

- **f_Servidores**

  Esta es una clase tipo formulario, donde se tiene que capturar el id del cliente propietario del servidor visualizado por la razón social, nombre del vps, cpu, ram, ips, disco duro, ancho de banda, sistema operativo, base de datos, ip publica, ip privada y una breve descripción del servicio, el formulario manda a validad los datos capturados por medio del componente br_Servidores.

- **br_Servidores**

  Es un componente que se encarga de manejar todas las reglas de negocio del módulo servidores, para permitir alojar la información de manera correcta en el arreglo de datos.

- **f_Gerentes**

  Esta es una clase tipo formulario, donde se tiene que capturar el nombre, correo electrónico, puesto, teléfono celular y de oficina, esto para poder complementar los campos requeridos en el módulo contratos. El formulario manda a validad los datos capturados por medio del componente br_Gerentes.

- **br_Gerentes**

  Es un componente que se encarga de manejar todas las reglas de negocio del módulo gerentes, para permitir alojar la información de manera correcta en el arreglo de datos.

3.4.2 Diseño de la base de datos

El diagrama de entidad asociación, el cual muestra las tablas utilizadas más relevantes, así como su asociación entre ellas, se muestra en la figura 3.6.

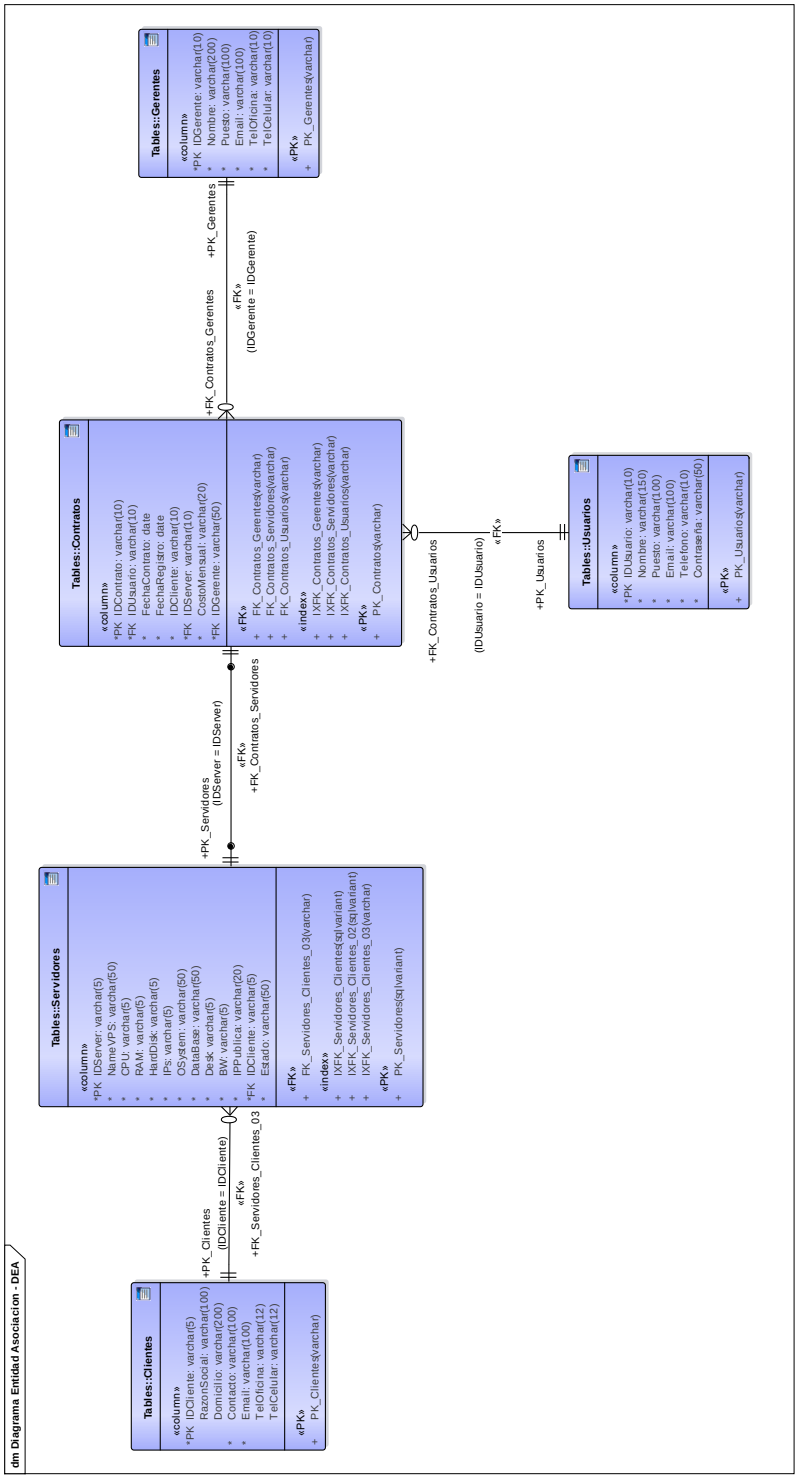

Figura 3.6 Diagrama de entidad asociación (DEA).

3.4.3 Diagrama de paquetes

En el diagrama de paquetes o diagrama de distribución se muestran las partes del sistema de lo que se debe de implementar en cada entidad (ver figura 3.7).

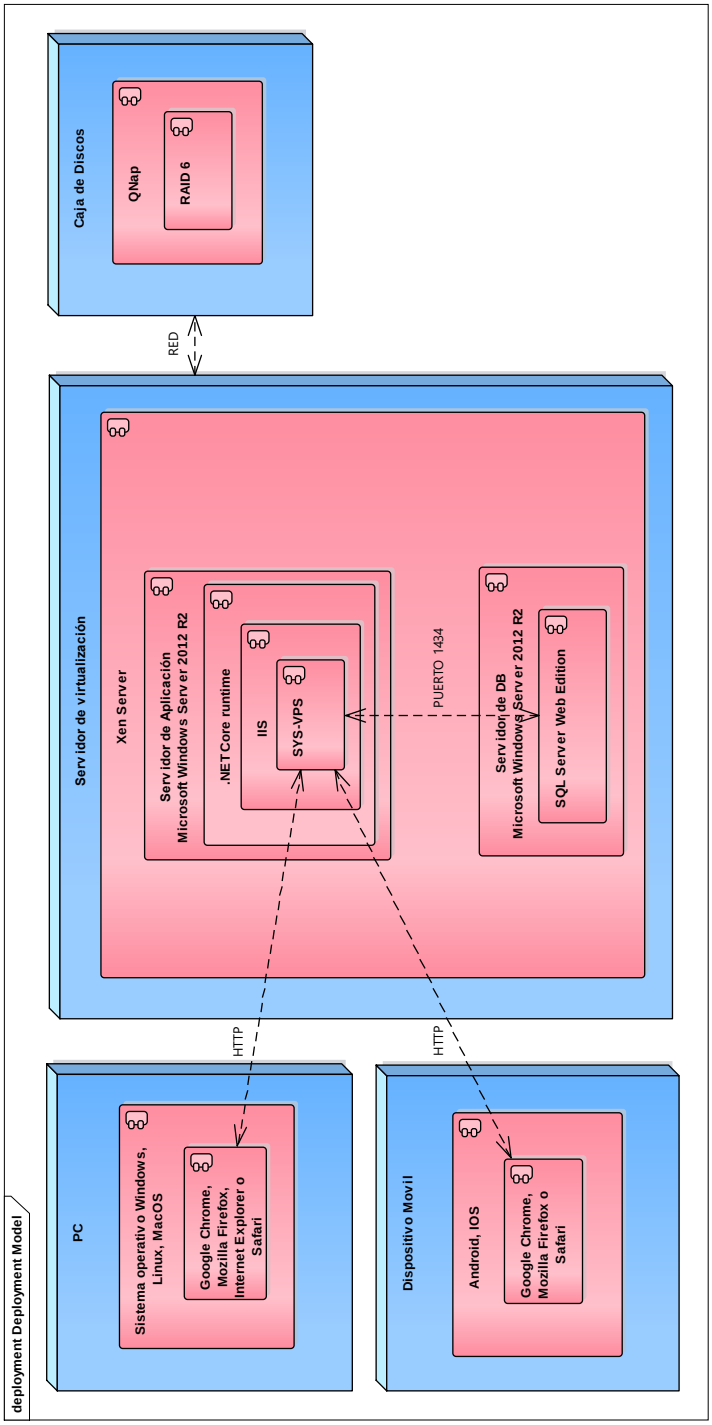

Figura 3.7 Diagrama de distribución de SYS-VPS.

- **PC**

  Es una computadora de escritorio en la cual se manipulará la herramienta SYS-VPS, por medio de un navegador web, ya sea Chrome, Firefox o Safari.

- **Dispositivo Móvil**

  Es un equipo portátil de fácil acceso como podría ser un teléfono celular inteligente o una tableta digital por el cual atreves de un navegador web pueda manipularse.

- **Servidor de virtualización**

  Servidor DELL que cuenta con un amplio poder de procesamiento para realizar la virtualización de servidores, llamados servidores virtuales privados, en los cuales serán alojadas nuestras herramientas.

  - **Servidor de aplicación**

    En este servidor se tiene habilitado el *Internet Information Services* (IIS) como servidor web, el cual se encargará de administrar los componentes de la herramienta **SYS-VPS**.

    En este servidor se tiene instalado el sistema operativo Windows Server 2012 R2.

  - **Servidor de BD**

    En este servidor se tiene instalado el SQL Server Web Edition, el cual es el encargado de administrar y manipular los datos. En el servidor se tiene instalado el sistema operativo Windows Server 2012 R2.

- **Caja de discos**

  Servidor de almacenamiento de datos donde son alojadas las imágenes creadas a partir de los servidores virtuales privados.

3.4.4 Arquitectura

Para la herramienta **SYS-VPS**, será una solución donde vamos a tener 3 proyectos, un proyecto llamado *Datos*, un proyecto llamado *Entidad*es y un proyecto llamado *Web*.

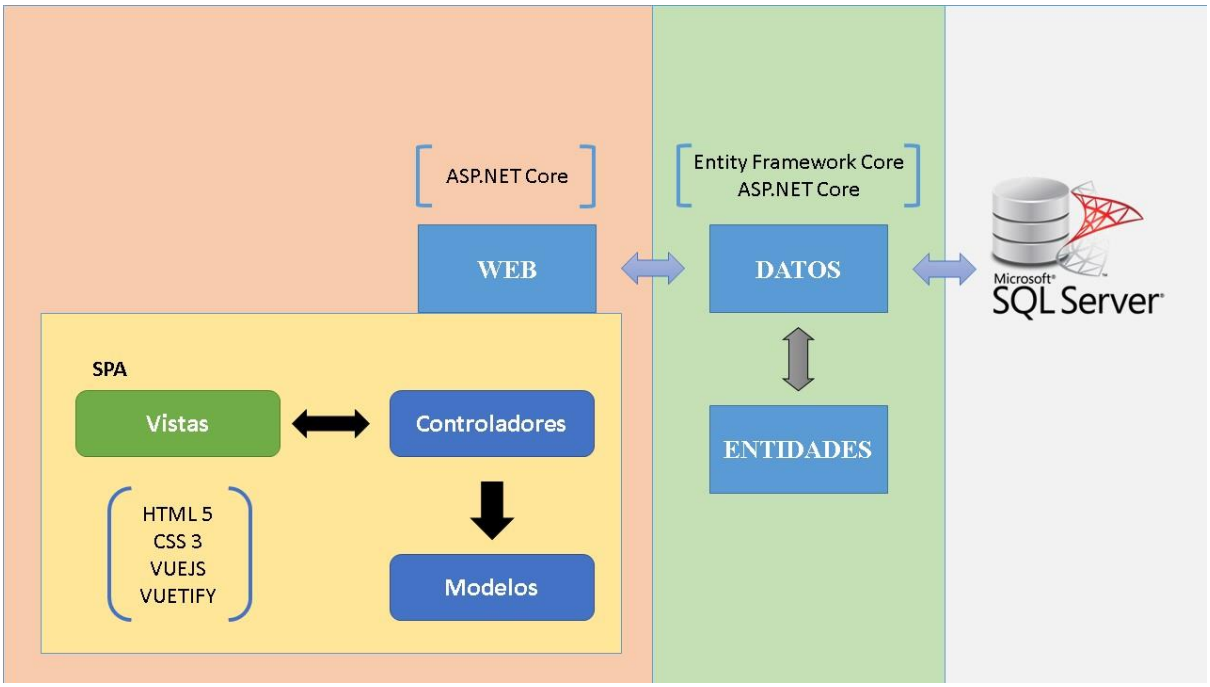

Figura 3.8 Arquitectura herramienta SYS-VPS

El gestor de base de datos a emplear en el desarrollo de este proyecto será SQL Server, se puede utilizar SQL Server 2008 en adelante hasta la versión más reciente, la versión que utilizaremos es la de SQL Server 2017.

En el proyecto Datos, está será la capa que interactúe con la base de datos, aquí estarán las referencias a Entity Framework y la referencia al gestor de base de datos con el que se va a trabajar. Para nuestro proyecto utilizaremos la referencia a SQL Server.

Entidades, en esta capa se encuentran las clases que definen el modelo de datos, aquí tendremos una clase por cada tabla de la base de datos.

Por último nuestro proyecto Web, en esta capa tendremos todo nuestro proyecto web utilizando el patrón MVC. Este patrón arquitectónico separa una aplicación en tres grupos principales de componentes, que son los modelos, las vistas y los controladores, permitiendo la separación de las preocupaciones. Con este patrón las solicitudes de los usuarios se enrutan en un controlador que es responsable de trabajar con el modelo para realizar acciones de usuario y/o recuperar resultados de consultas. El controlador elige la vista que le mostrará al usuario y le proporciona los datos del modelo que requiere.

Las vistas trabajarán con SPA (Aplicación de una Sola Página, in ingles Single Page Application). La vista será una sola página web con el propósito de dar una experiencia más fluida a los usuarios, Para esto utilizaremos VueJS y Vuetify que es un framework progresivo de componentes para VueJS sobre Material Design.

Se desarrollará una herramienta web con una arquitectura robusta, crearemos nuestra solución dividida en tres proyectos. Para el *backen*d de nuestra solución utilizaremos *ASP*.*NET Core MVC* y *Entity Framework Core*.

En nuestro *Frontend* se trabajará un SPA utilizando VueJS y el framework Vuetify, trabajaremos con el gestor de base de datos SQL Server, para la gestión de accesos y autorización utilizaremos Jason web tokens. Diseñaremos nuestra plantilla con Vuetify.

Nuestro *Backend* lo vamos a manejar con nuestro IDE Microsoft Visual Studio, dentro de nuestra solución tendremos el proyecto datos donde realizaremos el manejamiento de datos de nuestra base de datos. Tendremos también nuestro proyecto entidades con las clases que van a representar a cada una de las tablas de la base de datos y tendremos nuestro proyecto web, donde aplicaremos el patrón MVC. Tenemos aquí los controladores, los modelos y también tenemos es

este caso las vistas de nuestro proyecto. Pero las vistas las trabajaremos en este caso utilizando VueJS.

Para el *Frontend* implementaremos *SPA* utilizando *VueJS*, aquí tendremos todos los componentes que hacen peticiones *Ajax* mediante *axios* a nuestro *backend* (Todo esto para tener un código mejor ordenado).

Capítulo 4 Desarrollo

En este capítulo se detalla el proceso de desarrollo de herramienta web **SYS-VPS**, con base al análisis y diseño del sistema presentado en el capítulo anterior. El desarrollo se dividirá en 8 etapas, donde en la primera etapa se presenta la creación de la base de datos en nuestro gestor **SQL Server**. En la segunda etapa se verán las herramientas necesarias para desarrollar la aplicación, realizaremos la configuración previa de nuestro IDE y la creación de la solución en el mismo, en la tercera etapa se presentará la implementación del proyecto **Entidades** a la solución, generando una clase de entidad por cada tabla existente en la base de datos, en la cuarta etapa se agregará el proyecto **Datos** dentro de la solución y generar el mapeo de cada una de las tablas de la base de datos, procediendo con la quinta etapa donde se creará el proyecto **Web**, proyecto en el que implementaremos el patrón **MVC (Modelo-Vista-Controlador del inglés Model-View-Controller)**, desarrollando los módulos necesarios para la creación de cada uno de los controladores. Para la sexta etapa desarrollaremos las vistas de nuestra **SPA** utilizando el Framework **Vuetify**. En la séptima etapa realizaremos la gestión, acceso y autorización de usuarios, etapa en la que delimitaremos los permisos de acceso a la herramienta mediante los diferentes roles. Por último, en la octava etapa se realizará la creación del servicio de correo electrónico para generar las notificaciones vía correo electrónico.

4.1 Herramientas de desarrollo utilizadas

Las herramientas que fueron utilizadas para desarrollar la herramienta web fueron las siguientes:

- Visual Studio 2019: Es el Entorno de Desarrollo Integrado del acrónimo en inglés IDE que se utiliza por defecto para desarrollar aplicaciones orientadas al ecosistema .NET, trabaja con los lenguajes de programación de Microsoft, como: **C#**, Visual Basic, XAML y

tecnologías como el **.NET Framewor**k y el **ASP.NET Core**, este último utilizado para desarrollar la aplicación software del sistema SYS-VPS.

- Fiddler 4: Es una herramienta gratuita de monitorización de tráfico de PC, MAC, Linux y dispositivos móviles utilizada para depurar peticiones web actuando como servidor proxy entre tu ordenador e internet.
- Visual Studio Code: Es un editor de código fuente desarrollado por Microsoft para Windows, Linux y macOS. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es personalizable, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias. Es gratuito y de código abierto, editor en el cual instalaremos **VueJS** y **Vuetify** para el desarrollo de nuestro **front-end** de nuestra herramienta.

4.2 Creación de tablas de la base de datos

Una vez diseñados todos los diagramas de la herramienta web se procede a la siguiente fase de desarrollo, donde iniciaremos con la creación de la base de datos que es parte esencial de este proyecto ya que en ella se contendrá la información para el procesamiento de nuestra herramienta. Se optó por el gestor SQL Server ya que cuenta con una actualización más constante de seguridad, es uno de los gestores más populares en el desarrollo de aplicaciones y debido a que la empresa BTU Comunicación actualmente lo utiliza en otras de sus aplicaciones, de esta forma se verían facilitados los trabajos a futuro dentro de esta organización.

4.2.1 Instalación SQL Server

Antes de instalar nuestro motor de base de datos debemos conocer que existen varias versiones del mismo. La versión que utilizaremos durante el desarrollo de este proyecto será la **Express**, ya que es una edición gratuita de SQL Server ideal para el desarrollo y la producción de aplicaciones de escritorio, aplicaciones web y pequeñas aplicaciones de servidor.

Una particularidad de utilizar esta versión es que nuestro tamaño máximo de base de datos es de 10 GB, constante que no es ningún impedimento para utilizar esta versión, ya que no almacenaremos ningún otro tipo de dato que no sea texto, provocando que el tamaño de nuestra base de datos sea muy pequeño en comparación con el límite.

Para obtener **SQL Server Express** debemos acceder a su sitio oficial, una vez descargado el instalador y ejecutado nos mostrara la siguiente ventana:

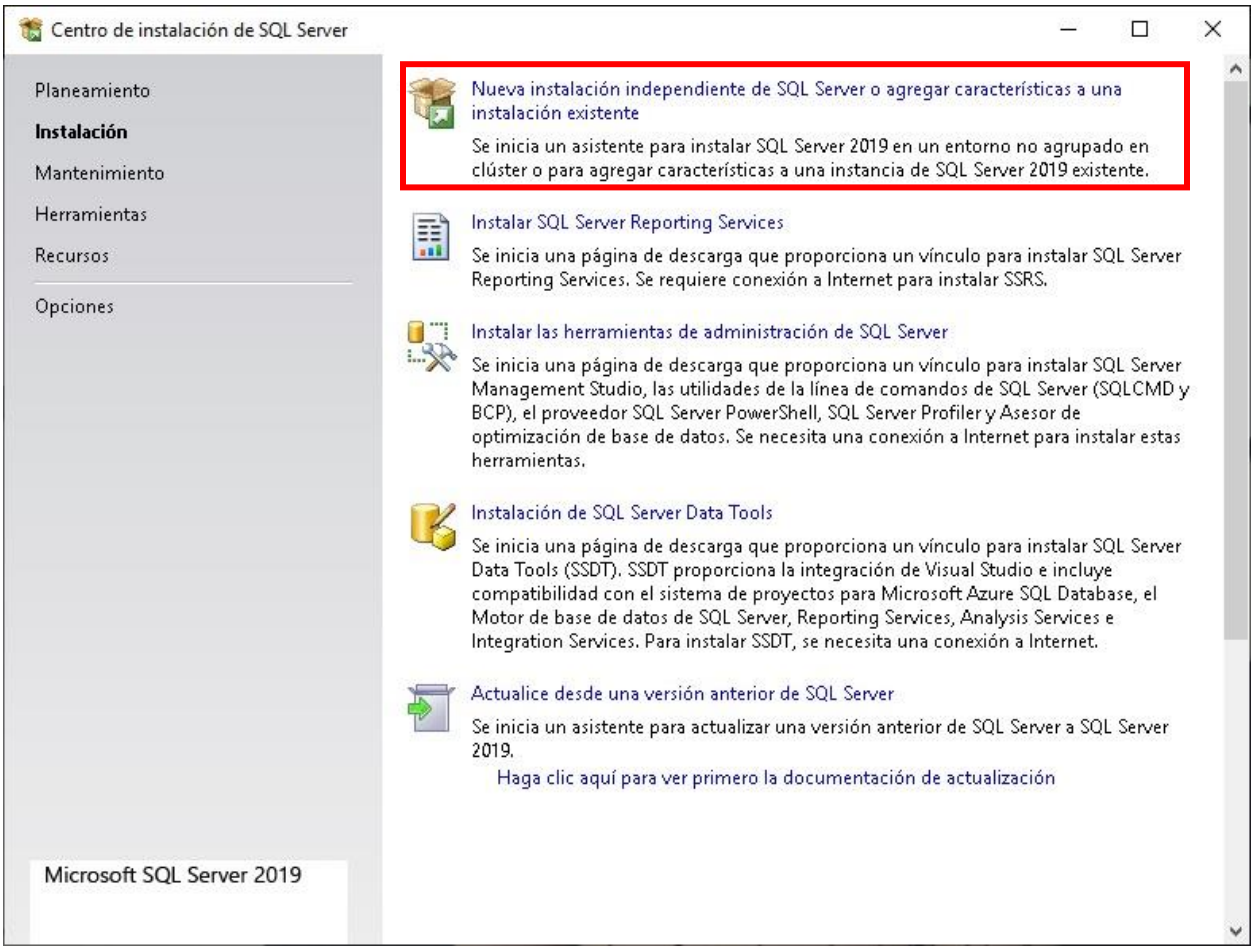

Figura 4.1 Centro de instalación de SQL Server

Daremos clic justamente en “Nueva instalación independiente de SQL Server…” (ver figura 4.1), entonces comenzaremos la descarga del motor SQL y su instalación, la instalación es rudimentaria a otro tipo de aplicaciones, aunque hay un conjunto de configuraciones que necesitamos realizar antes para poder trabajar de forma correcta, las configuraciones son las siguientes:

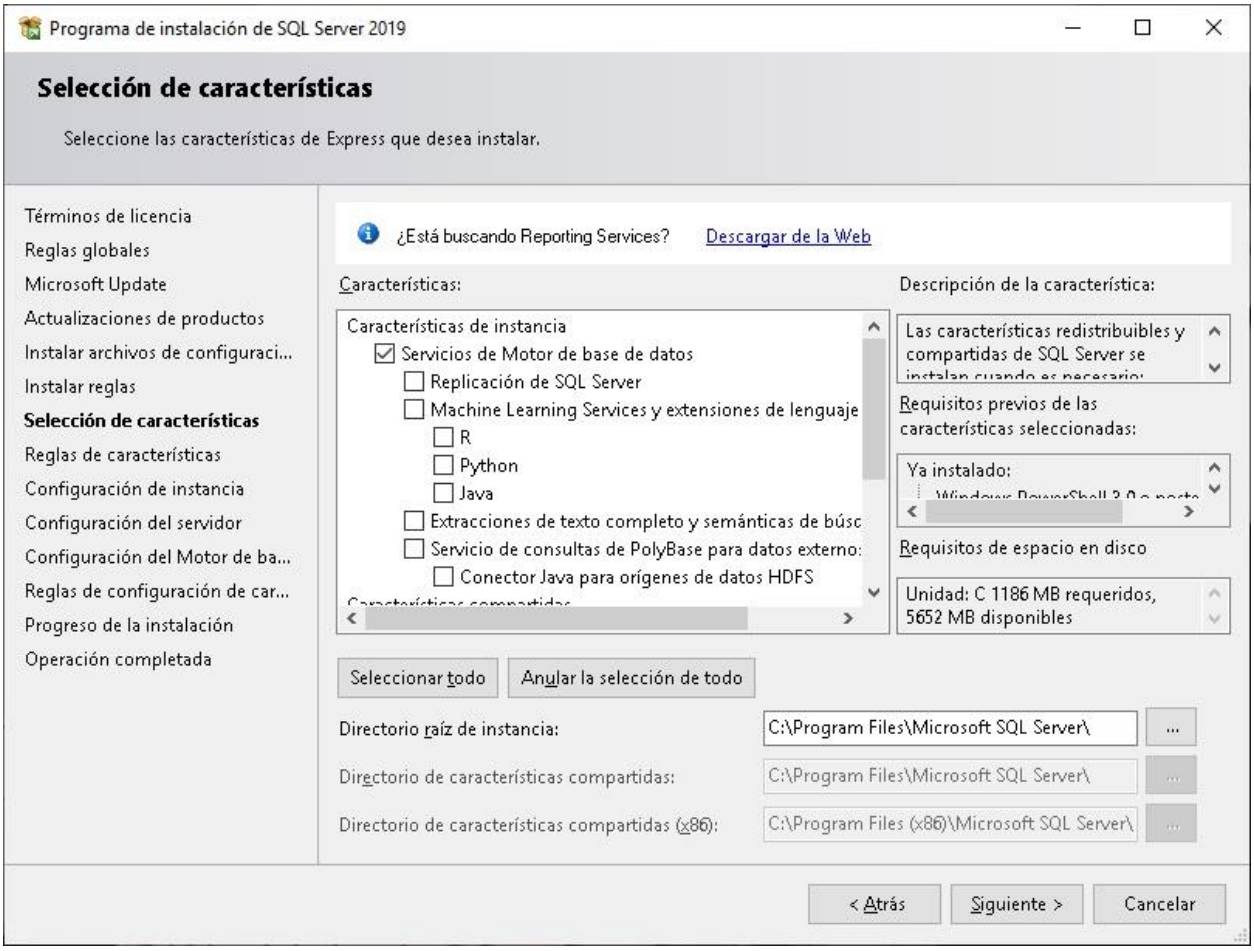

Figura 4.2 Selección de características

La **selección de características** es una de las partes más importantes durante la instalación, podemos establecer las que nos dejan por defecto y agregar las que consideremos necesarias, esta acción no siempre es la mejor recomendación ya que podría consumir un mayor número de recursos, las características que necesitamos para el desarrollo de este proyecto son: (ver figura 4.2)

- Servicios de Motor de SQL Server
- Compatibilidad con versiones anteriores de las herramientas
- LocalDB

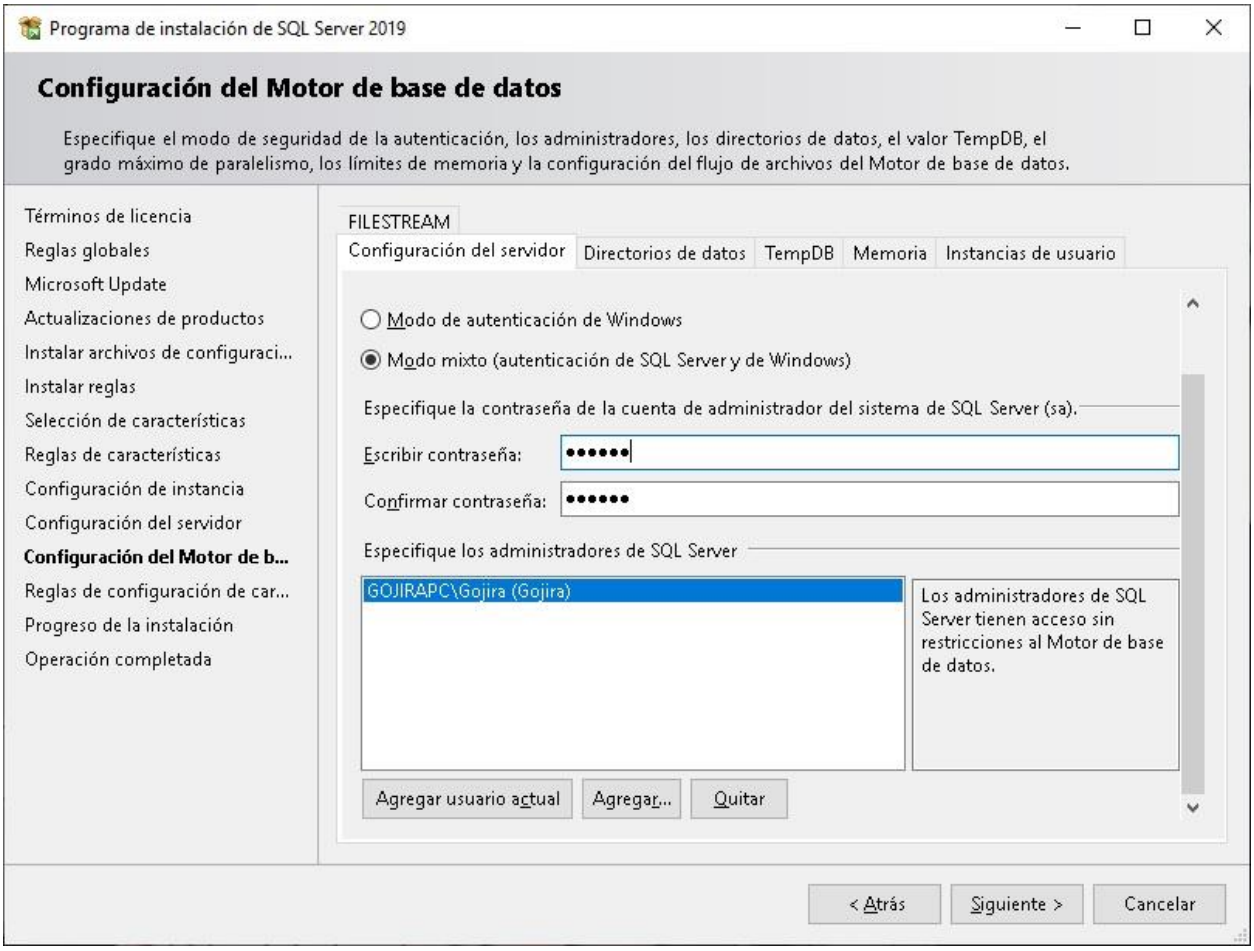

Figura 4.3 Configuración del Motor de base de datos.

Modo de autentificación, esta es una de las configuraciones en la que tenemos que realizar una modificación, por defecto solo tenemos la autentificación de Windows, esta autentificación utilizara nuestro inicio de sesión de Windows para poder conectarse, en este caso utilizaremos el modo mixto para autentificarnos, Autentificación de SQL Server y de Windows (ver figura 4.3), en caso de que no aparezca nuestro usuario para ser autentificado pulsaremos el botón “Agregar usuario actual”.

Una vez realizadas estas configuraciones podremos trabajar de forma correcta con nuestro motor de base de datos. Terminada la instalación se nos preguntará si deseamos instalar el **MSSM**S

(Microsoft SQL Server Management Studio), a lo cual marcaremos esta casilla e instalaremos, esto con el propósito de obtener una interfaz que nos facilite la creación de nuestra base de datos.

4.2.2 Acceso y Creación de la base de datos

Una vez instalado nuestro MSSMS intentaremos conectarnos a nuestro servidor de base de datos, como visualizamos en la ocasión anterior, al momento de hacer la instalación configuramos que podríamos acceder al servidor con la autentificación de Windows y con el usuario SQL Server.

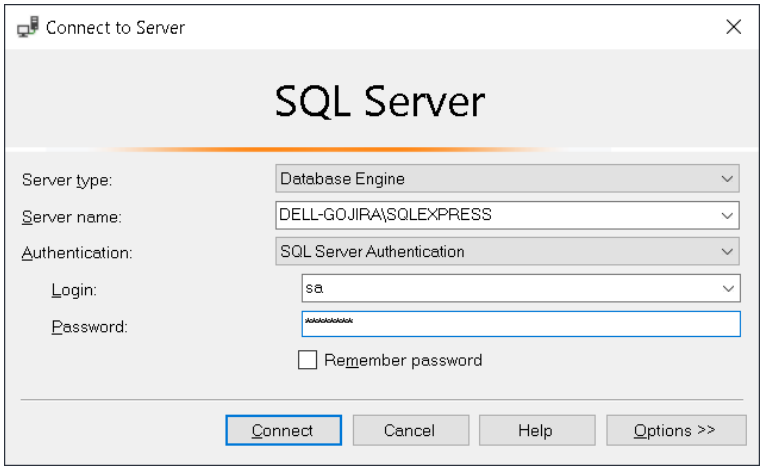

Figura 4.4 Conectarse al Servidor

Para conectarse al servidor podemos elegir el modo de autentificación, para acceder utilizando la autentificación de SQL Server necesitaremos escribir el nombre de usuario "**sa**" (ver figura 4.4). Por defecto el gestor no nos indica este nombre de usuario ya existente.

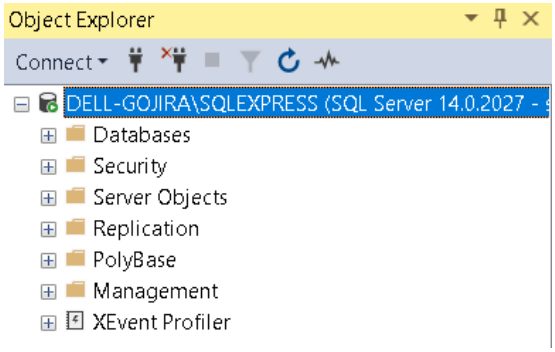

Figura 4.5 Conexión exitosa.

Una vez comprobado el acceso de ambas formas procederemos con la creación de nuestra base de datos, para esto daremos clic en el botón "**New Query**" de nuestro Management Studio, desplegándonos un área de trabajo como se muestra en la figura 4.6.

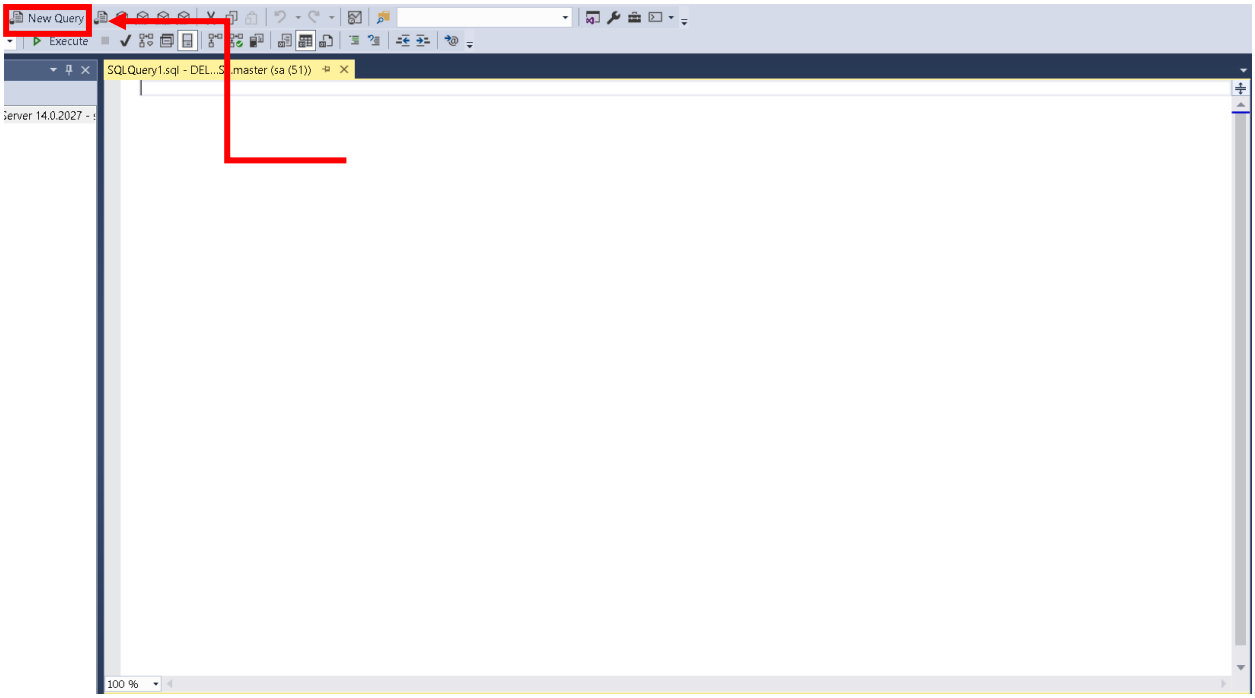

Figura 4.6 New Query.

Para crear nuestra base de datos ejecutaremos el siguiente código: (ver figura 4.7)

```
create database DBSYSVPS;
```

Figura 4.7 Código para la creación de la base de datos.

Seguido de esto crearemos nuestras tablas donde almacenaremos la información que nuestra herramienta utilizara. Antes de esto debemos indicarle a nuestro gestor que utilice la base de datos de nuestro proyecto. Esta acción podremos realizarla utilizando el siguiente código (ver figura 4.8).

```
use DBSYSVPS;
```

Figura 4.8 Código para seleccionar la base de datos.

Hecho esto, crearemos las tablas de nuestro sistema, tablas que han sido representadas en nuestro diagrama de entidad asociación (ver figura 3.6), para generar algunas de estas ejecutaremos el siguiente código.

```
create table vmclient (
    idclient integer primary key identity,
    clientname varchar(50) not null unique,
    clientfullname varchar(256) null,
    clientemail varchar(100) null,
    clientphone varchar(20) null,
    clientcontact varchar(100) null,
    emailcontact_tecnico varchar(100) not null
    estado bit default(1)
);
create table networkbond (
    idnw integer primary key identity,
    nwbond varchar(100) not null,
    nwestado bit default(1)
);
create table osfamily (
    idos integer primary key identity,
    osfamilyname varchar(100) not null,
);
create table osversion (
    idversion integer primary key identity,
    idos integer not null,
    osversion varchar(50) not null unique,
    descripcion varchar(256) null,
    FOREIGN KEY (idos) REFERENCES osfamily(ido
);
create table pools (
    idpool integer primary key identity,
    poolname varchar(50) not null,
    pooldescripcion varchar(256) null,
    poolestado bit default(1)
);
```

```
create table vps (
    idvps integer primary key identity,
    idclient integer not null,
    vmname varchar(50) not null,
    vm_uuid varchar (256) not null,
    vcpus int not null,
    ram int not null,
    hdisk integer not null,
    bandw integer null,
    idnw integer not null,
    idos integer not null,
    idversion integer null,
    idsql integer null,
    idsqlversion integer null,
    internal_ip varchar(256) not null,
    external_ip varchar(256) not null,
    createon date not null,
    idusuario integer not null,
    dnsname varchar(256) null,
    idvmtype integer not null,
    idpool integer not null,
    notes varchar(256) null,
    rmtaccesssal integer null,
    estado bit default(1),
    FOREIGN KEY (idclient) REFERENCES vmclient(idclient),
    FOREIGN KEY (idnw) REFERENCES networkbond(idnw),
    FOREIGN KEY (idos) REFERENCES osfamily(idos),
    FOREIGN KEY (idversion) REFERENCES osversion(idversion),
    FOREIGN KEY (idsql) REFERENCES sqlfamily(idsql),
    FOREIGN KEY (idsqlversion) REFERENCES sqlversion(idsqlversion)
    FOREIGN KEY (idusuario) REFERENCES usuario(idusuario),
    FOREIGN KEY (idvmtype) REFERENCES vmtype(idvmtype),
    FOREIGN KEY (idpool) REFERENCES pools(idpool)
);
```

Figura 4.9 Código para generar tablas mediante SQL.

Hecho esto de forma satisfactoria podemos pasar a la siguiente etapa de desarrollo, creación del **Back-end** de nuestra aplicación web.

4.3 Creación de la solución y Proyecto Entidades

Como se especificó en el capítulo 2 el IDE utilizado para el desarrollo de la herramienta es Microsoft Visual Studio, para descargarlo solo tendremos que acceder a la siguiente liga https://visualstudio.microsoft.com/es/downloads/, después de acceder nos aparecerán todas las versiones disponibles de Visual Studio 2019 procederemos a descargar la versión **community**, el cual es un IDE gratuito para estudiantes, desarrolladores de código abierto y desarrolladores individuales.

La descarga comienza de forma automática después de haber seleccionado la versión community, de no ser así la página nos proporcionara un link el cual podemos utilizar en caso de que lo anterior mencionado no suceda.

El proceso para instalar el IDE es similar a la de cualquier programa, lo fundamental es seleccionar los **paquetes de desarrollo** necesario para poder trabajar en la elaboración de nuestra herramienta web.

Los componentes necesarios para el desarrollo correcto son los siguientes:

- Universal Windows Plataform
- .Net desktop development
- ASP.NET and web development
- .NET Core cross-plataform development

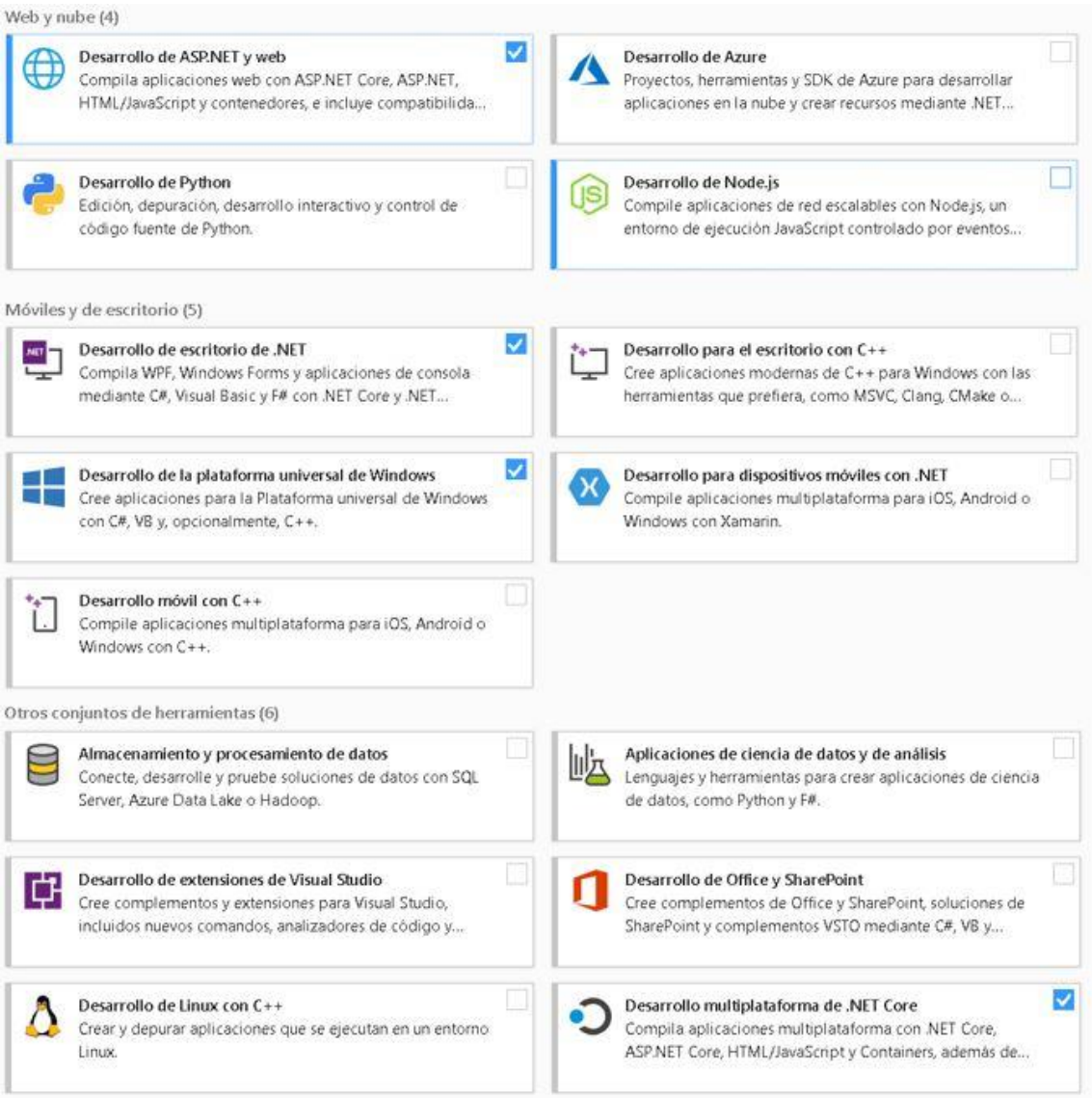

Figura 4.10 Componentes de desarrollo Visual Studio

Finalizada la descarga e instalación de los componentes es importante asegurarnos que todo se haya instalado de manera correcta. Después de haber realizado la instalación es recomendable reiniciar su equipo para que se puedan hacer correctamente los ajustes en Visual Studio. A continuación, se describirá el proceso de la creación del proyecto.

4.3.1 Creación de la solución

Cuando ejecutamos Visual Studio nos proporcionará opciones para trabajar en un proyecto ya existente (ver figura 4.11), para el desarrollo de la herramienta se creó un proyecto nuevo, para ello se seleccionó la opción “Crear un proyecto”.

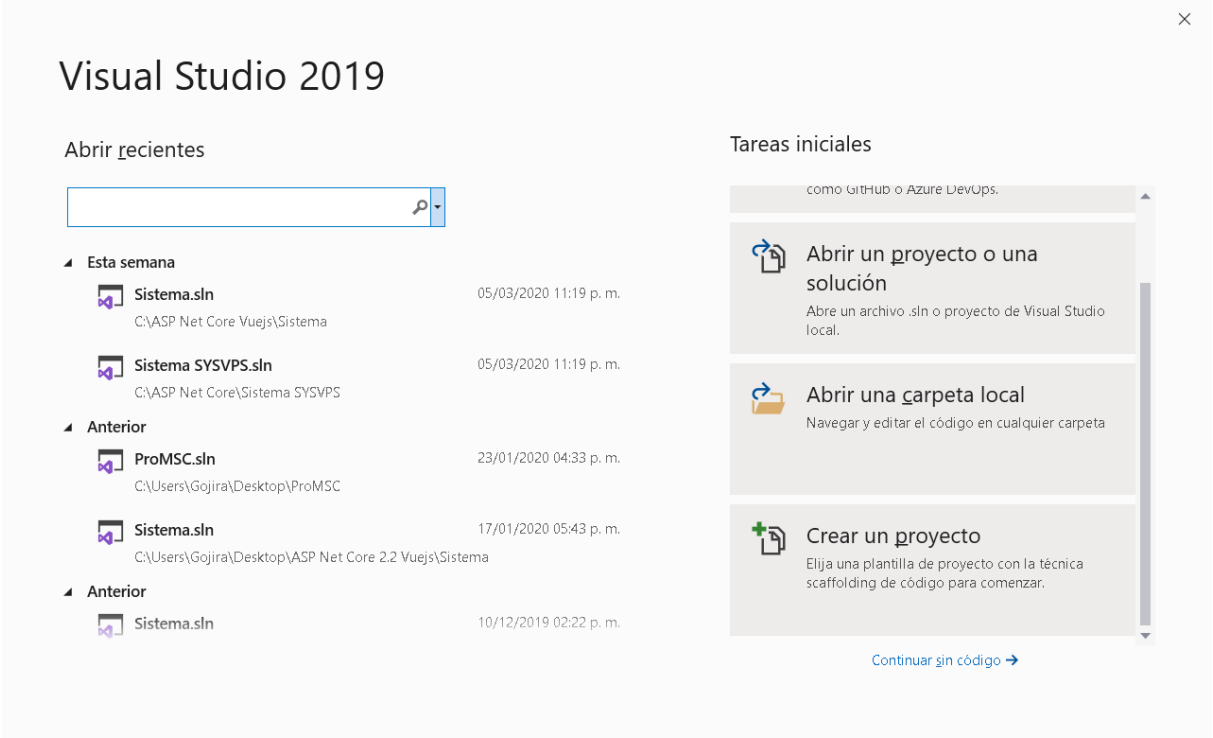

Figura 4.11 Asistente de inicio de Visual Studio

Después de haber elegido esta opción de crear un nuevo proyecto el asistente muestra una ventana donde permite crear diferentes tipos de proyectos dependiendo del tipo de aplicación que se desee generar, para el desarrollo de la herramienta se usará principalmente Solución en blanco (**Blank solution**) como se observa en la figura 4.12.

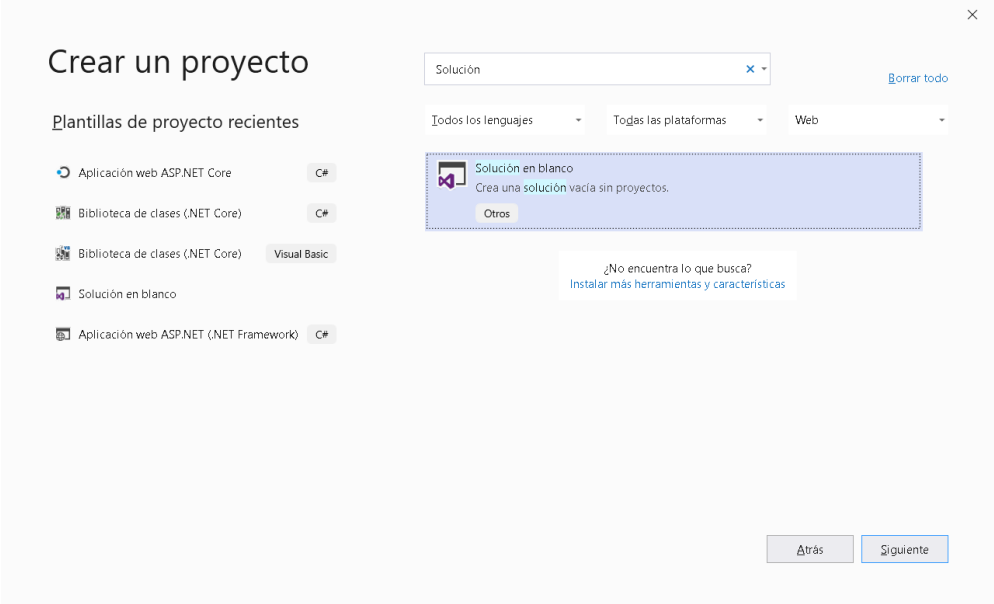

Figura 4.12 Ventana para seleccionar el tipo de proyecto con Visual Studio

Posterior a seleccionar el tipo de proyecto es necesario dar un nombre a la solución y asignar un directorio para la ubicación del mismo como se observa en la figura 4.13 y por ultimo dar clic en crear.

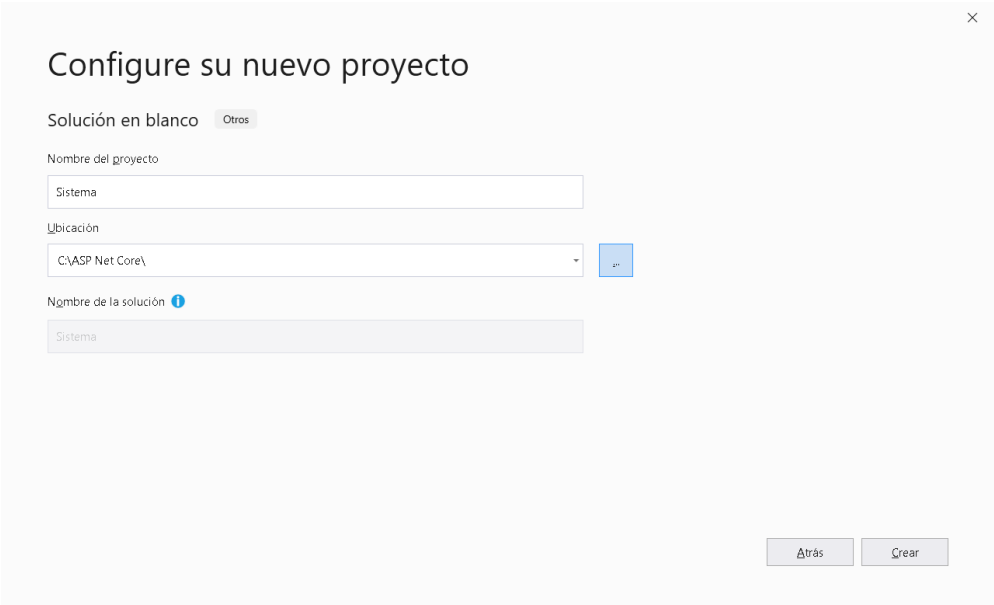

Figura 4.13 Ventana para asignar un nombre y directorio al proyecto.

Hecho esto se cerrará el asistente y nos mostrara el entorno de nuestro IDE, a partir de este momento podemos comenzar a crear los proyectos que intervendrán dentro de nuestra solución.

4.3.2 Creación del proyecto Entidades

Una vez creada nuestra solución procederemos a crear el proyecto Entidades, proyecto en el que crearemos una clase de entidad por cada tabla existente en la base de datos. Para realizar esto en nuestra solución vacía daremos clic derecho, nos posicionaremos en la opción agregar y daremos clic en agregar un nuevo proyecto como se muestra en la figura 4.14.

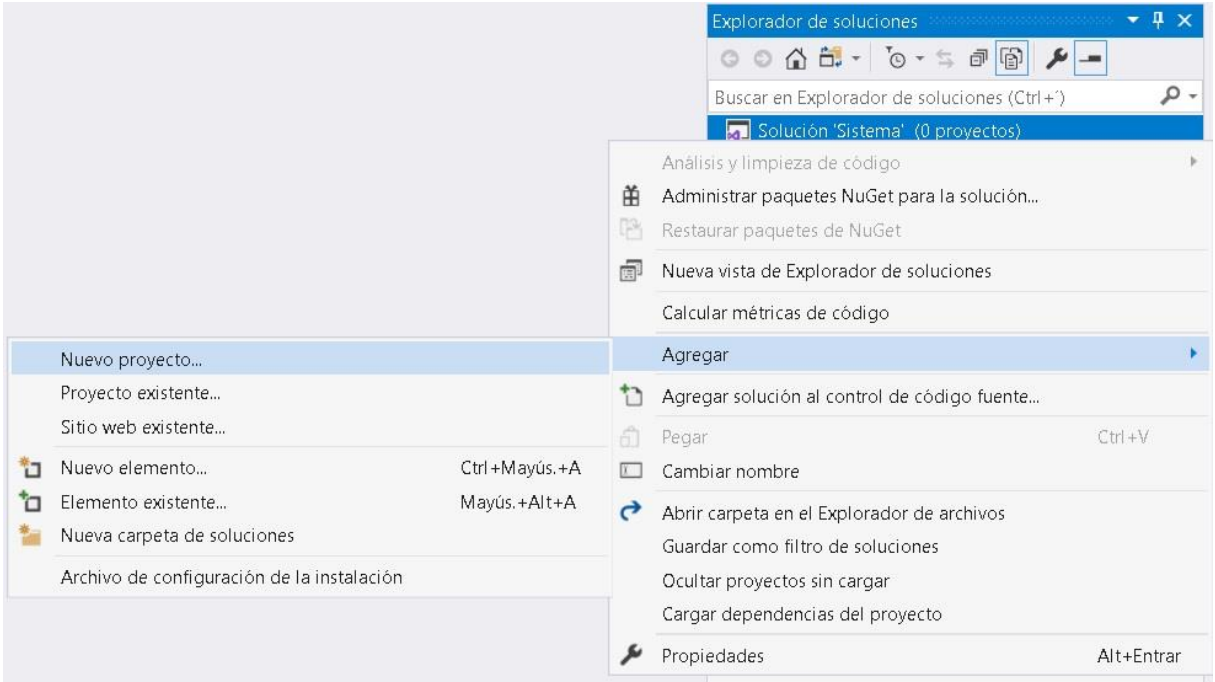

Figura 4.14 Agregar proyecto a nuestra solución en Visual Studio.

Después de hacer esto nuestro IDE nos mostrara una ventana de dialogo en donde seleccionaremos el proyecto que agregaremos a la solución, para nuestro proyecto Entidades elegiremos biblioteca de clases (**.NET Core**) con el lenguaje de programación C# como se muestra en la figura 4.15 y daremos clic en siguiente.

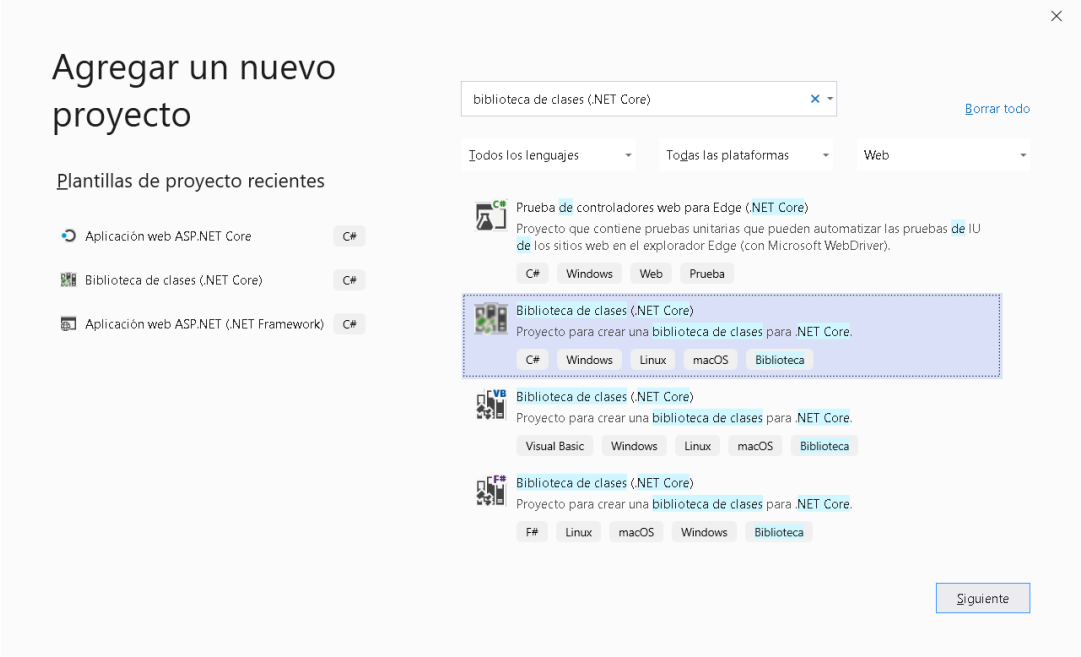

Figura 4.15 Proyecto Biblioteca de clases (.NET Core).

Una vez que hayamos dado clic en siguiente se nos solicitara proporcionar un nombre de proyecto, después de hacerlo daremos clic en Crear para finalizar con la creación del proyecto (ver figura 4.16).

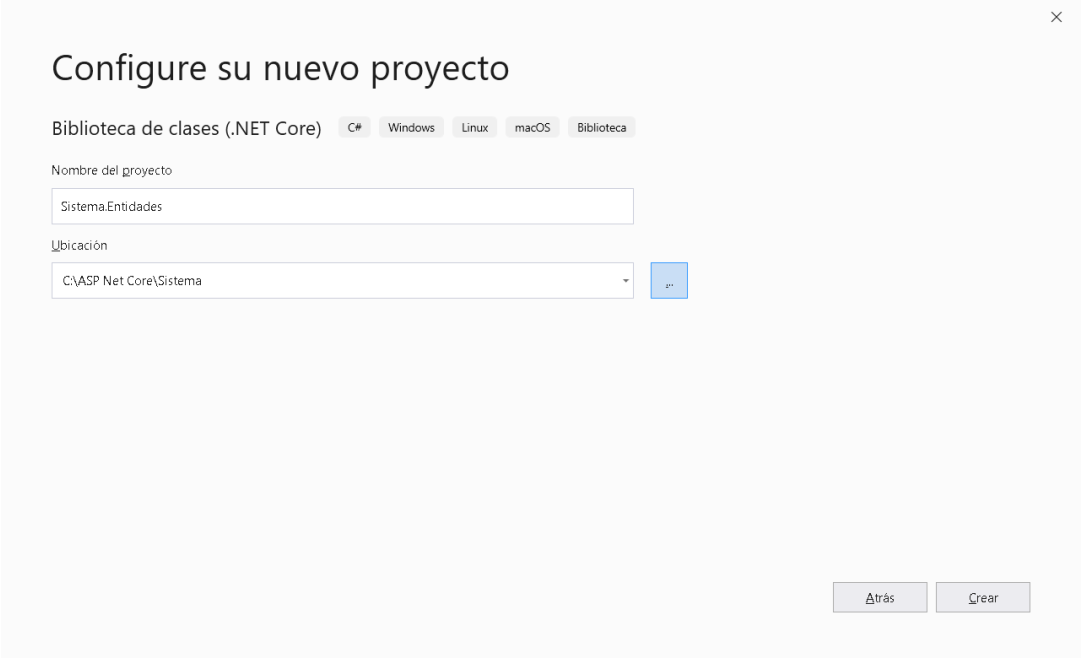

Figura 4.16 Ventana para proporcionar un nombre al proyecto.

Utilizando el mismo procedimiento anterior crearemos el proyecto Datos, para nuestro proyecto Web utilizaremos el tipo de proyecto **Aplicación web ASP.NET Core** como se muestra en la figura 4.17.

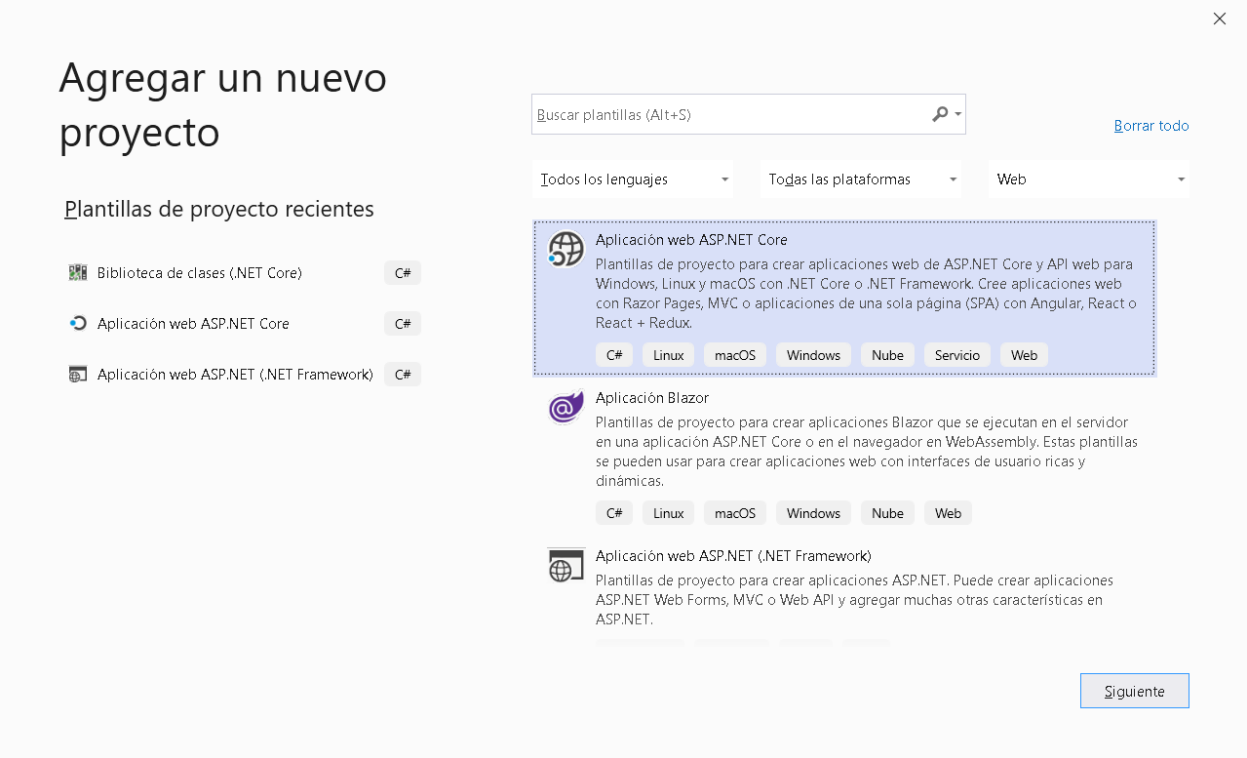

Figura 4.17 Proyecto Aplicación web ASP.NET Core.

El nombre que elegiremos para este proyecto será Web, hecho esto el asistente de Visual Studio muestra una ventana donde se puede trabajar con diferentes tipos de plantilla en una aplicación web ASP.NET Core, para el desarrollo de la herramienta se usará la plantilla **API (Interfaz de programación de aplicaciones)** esta plantilla proporciona un proyecto ya pre configurado con un controlador y sus vistas (ver ilustración 4.7). Con esto al dar clic Visual Studio preparará el entorno de trabajo para el desarrollo del proyecto. En este proyecto se trabajará con la versión 2.1 de **ASP. NET Core**, en este caso no es necesario instalar el **SDK** ya que los paquetes descargados durante la instalación de Visual Studio ya los tiene integrados y listos para trabajar.

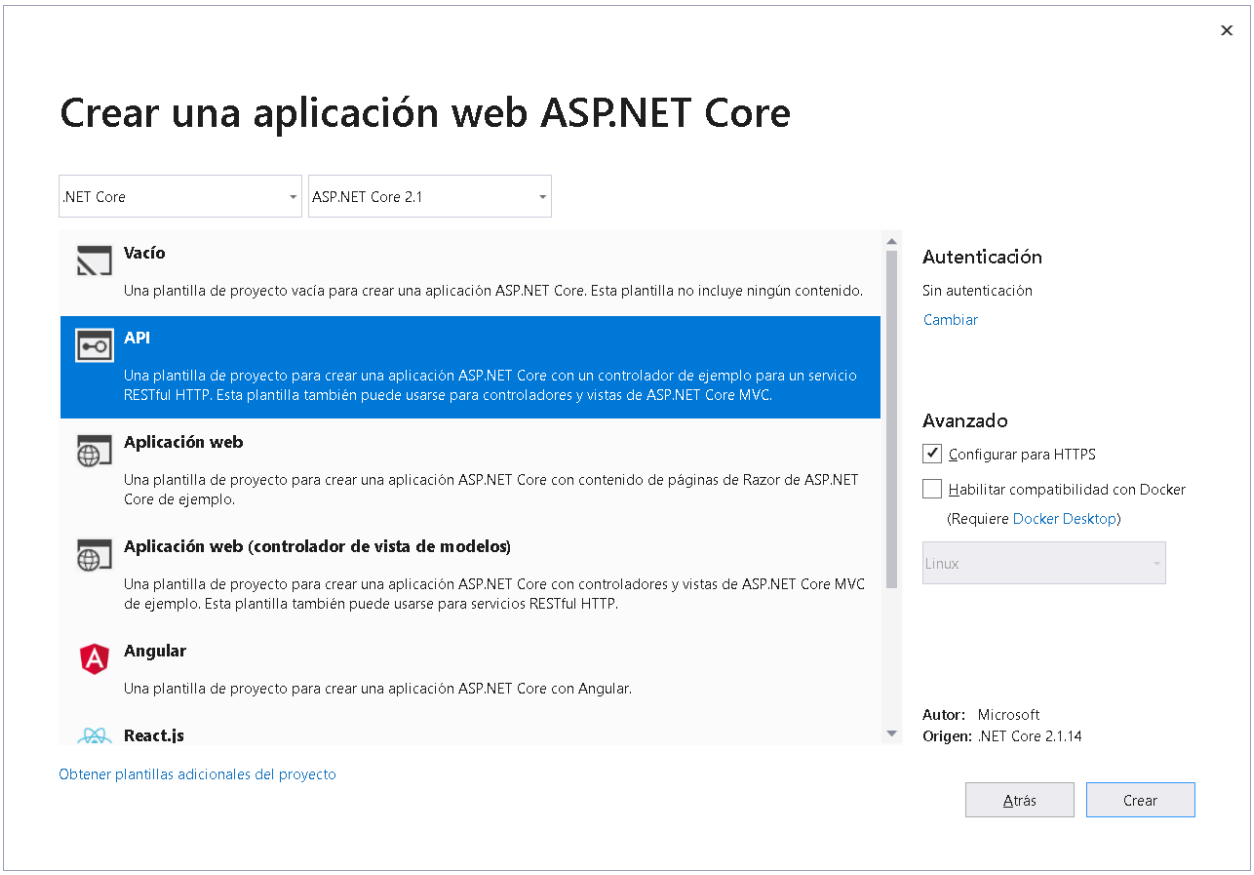

Figura 4.18 Ventana para la selección de la plantilla a trabajar.

4.4 Proyecto Entidades

En esta etapa tenemos nuestra solución en blanco, al generar el tipo de proyecto Biblioteca de clases nos genera de forma automática una clase, estas clases las eliminaremos del proyecto Entidades y el proyecto Datos, de la misma forma en nuestro proyecto Web nos genera un controlador de forma automática, procederemos a eliminarlo, de esta forma obtendremos un entorno más limpio en el cual trabajar.

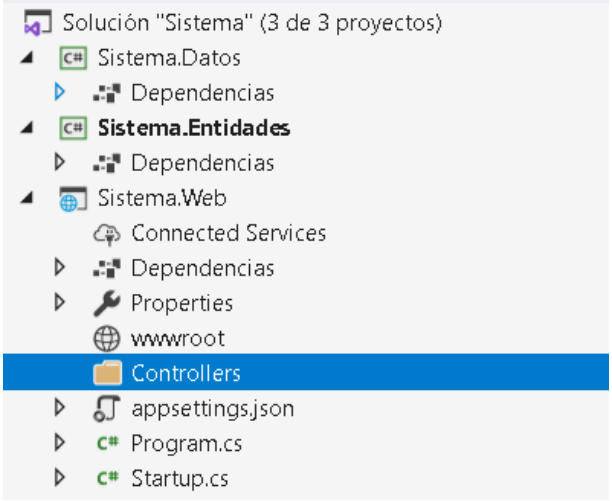

Figura 4.19 Solución de trabajo Visual Studio.

El propósito del proyecto entidades es crear una capa de abstracción entre la capa de datos y la capa del modelado de negocios, en este proyecto crearemos una clase de entidad por cada tabla existente en la base de datos, principalmente crearemos un grupo de carpetas en las cuales almacenaremos las clases que contengan una relación en su tipo de datos, esto para trabajar de una forma más ordenada, dentro de ellas crearemos las clases necesarias para trabajar con nuestra base de datos. Podremos observar lo anterior mencionado en la figura 4.20.

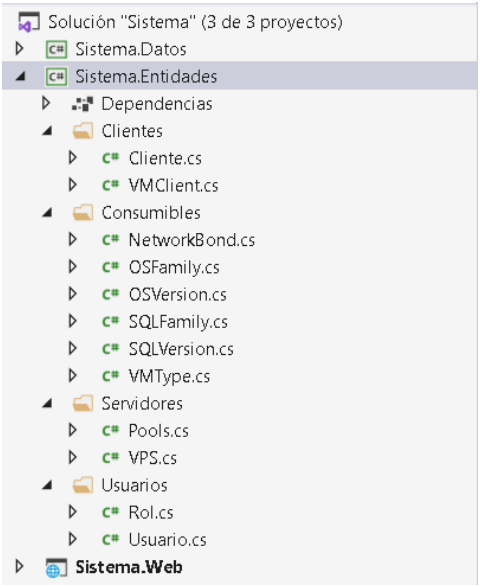

Figura 4.20 Proyecto Entidades.

4.4.1 Creación de las entidades

La capa de entidades contiene las clases que representan los modelos dentro de la **arquitectura MVC**, las cuales fueron creadas en base al diagrama de clases presentado en el capítulo 3. Estos datos son utilizados para la manipulación de información en tiempo de ejecución, además estos modelos son una representación de las tablas de la base de datos, entonces esto permite la comunicación entre un sistema orientado a objetos y un relacional como lo es nuestro **gestor SQL**. Comúnmente cuando se desarrolla una aplicación que tenga que almacenar información de forma permanente el primer paso es la creación de una base de datos y posterior el desarrollo de la aplicación la cual contendrá clases que representarán las tablas de la base de datos para almacenar la información temporalmente, con **EntityFramework** permite trabajar con una base de datos ya existente utilizando el enfoque **Database First**.

Con lo mencionado anteriormente primero se crearon las clases que se pueden observar en la figura 4.20, varios autores coinciden en que una buena clase de entidad debe ser puramente el reflejo de los datos que contiene la tabla sin agregar reglas de negocio, en estas clases podemos hacer uso de las **DataAnnotations**, que son validaciones de datos de acuerdo a nuestras necesidades, tomando el ejemplo de la figura 4.21, en este caso indicamos con DataAnnotations que podemos apreciar que son palabras escritas entre corchetes, en este caso la propiedad **clientname** estamos indicando que es un dato requerido con la propiedad **Required** y asignamos un valor máximo y minimo de caracteres que puede contener con la propiedad **StringLength(50, MinimumLength = 3)**, podemos apreciar el resto de validaciones con las que cuentan las tablas en el diagrama de entidad asociación encontrado en el capítulo 3.

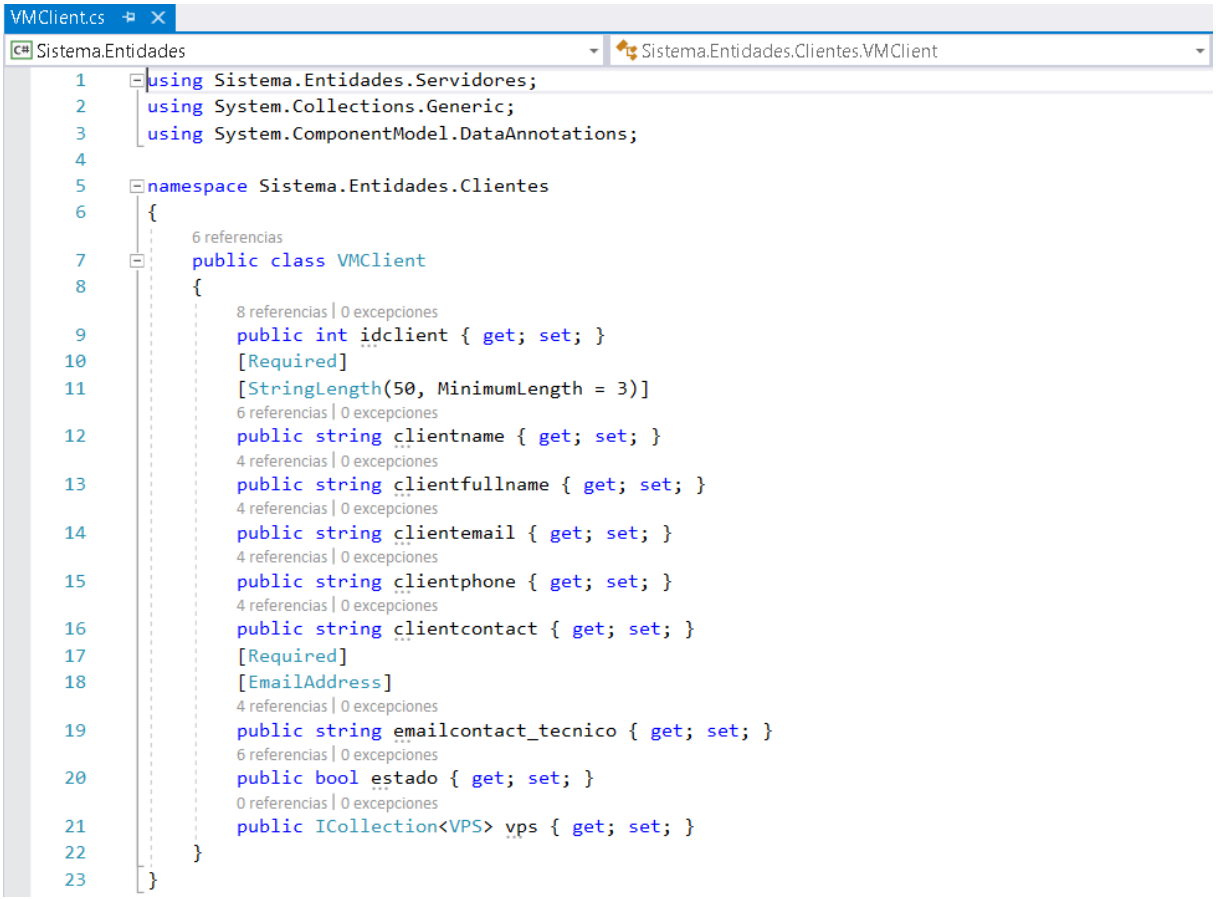

Figura 4.21 La clase VMClient y sus propiedades en C#.

Para indicarle a nuestro proyecto que una clase tiene relación con otra, es decir que son tablas que tienen relación en la base de datos se crean propiedades de navegación que hacen referencia a otras clases.

Figura 4.22 Propiedades para hacer relaciones entre clases.

En este caso en la entidad **VPS** se crearon nueve propiedades que hacen referencia a las entidades vmclient, networkbond, osfamily, osversion, sqlfamily, sqlversion, usuario, vmtype y pools, indicando de esta forma que un VPS solo puede tener una de estas clases (figura 4.22).

4.5 Proyecto Datos

Una vez creadas todas las entidades procederemos a agregar las dependencias **EntityFrameworkCore.SqlServer** y **EntityFrameworkCore.Tools** de Microsoft utilizando nuestro administrador de paquetes **NuGet**. Para realizar esto daremos clic derecho sobre nuestro proyecto Datos y seleccionaremos Administrador de paquetes NuGet, nos abrirá un panel en el cual en la pestaña Examinar buscaremos las dependencias necesarias (ver figura 4.23).

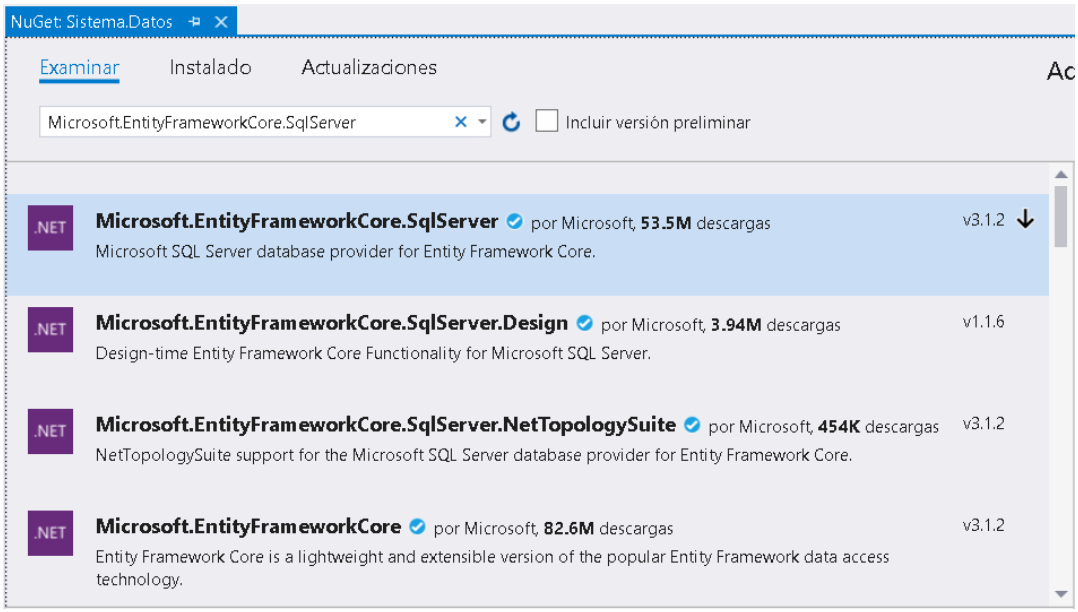

Figura 4.23 Búsqueda de paquetes NuGet.

Una vez encontrado procederemos a instalar el paquete a nuestro proyecto, seleccionaremos el paquete y elegiremos la versión del mismo, en este caso estamos utilizando el **SDK 2.1** de .Net Core, de esta forma elegiremos una versión compatible, en este caso será la versión 2.2.1 como se muestra en la figura 4.24.

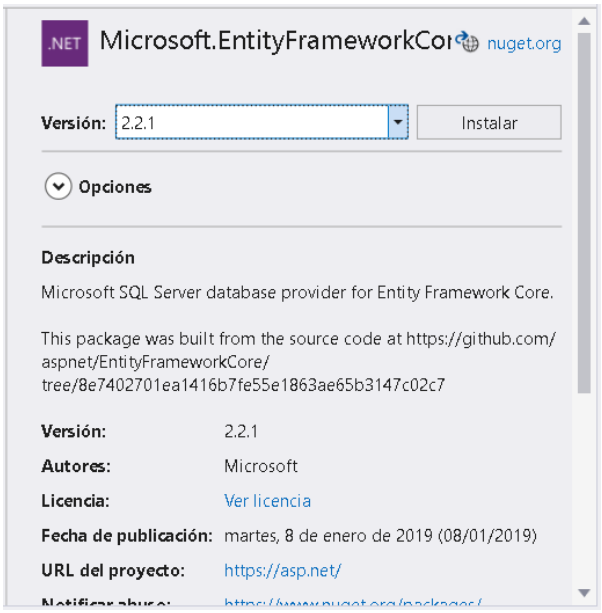

Figura 4.24 Instalar paquete NuGet.

Utilizando el mismo procedimiento instalaremos el paquete **EntityFrameworkCore.Tools** de Microsoft. Una vez instalados ambos paquetes crearemos nuestro contexto de datos, para poder realizar esto crearemos una carpeta dentro del proyecto Datos llamada **Mapping**, organizaremos nuestras clases de la misma forma que en nuestro proyecto Entidades para tener una solución ordenada (ver figura 4.25).

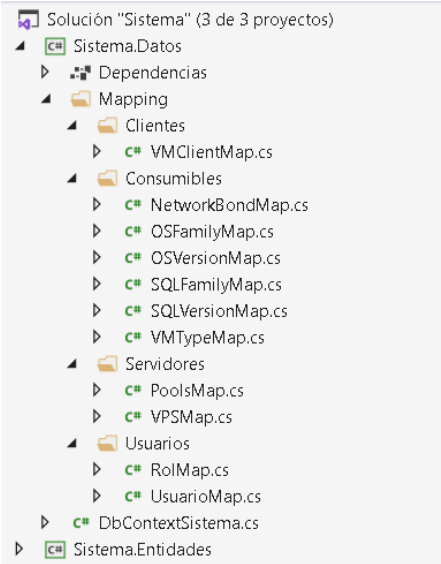

Figura 4.25 Carpeta Mapping.

Para crear nuestras clases debemos relacionar nuestro proyecto Datos con nuestro proyecto Entidades, para lograr esto debemos ir nuestro apartado dependencias dentro del proyecto datos y seleccionar agregar una dependencia, en este caso agregaremos a nuestro proyecto Entidades (ver figura 4.26).

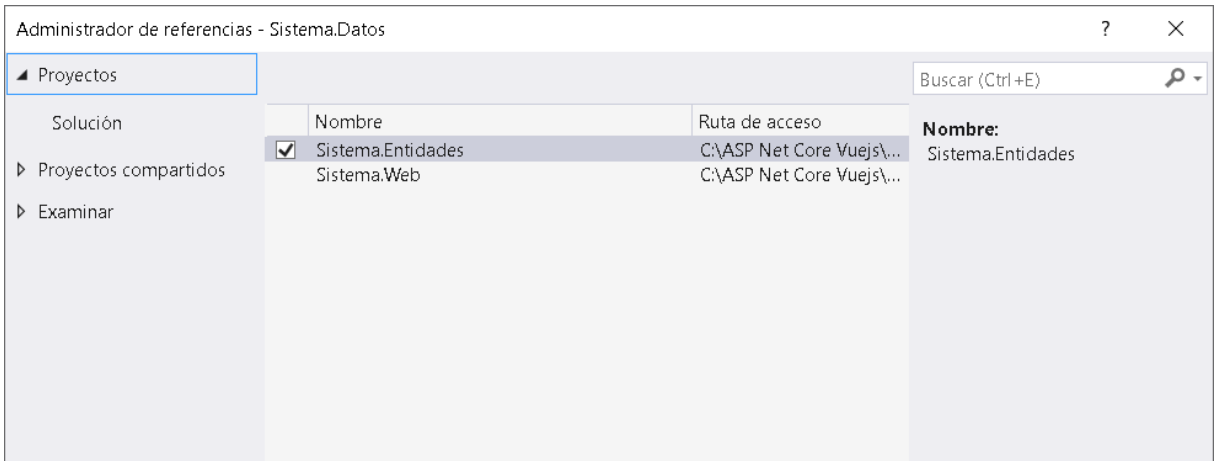

Figura 4.26 Administrador de referencias.

Una vez agregada nuestra referencia y paquetes Nuget, vamos a crear la clase que coordina la funcionalidad de **EntityFramework** para un modelo de datos dado, está es la clase de contexto de la base de datos y le asignaremos como nombre **DbContextSistema**, para realizar esto agregaremos una clase que haremos pública que va a heredar de la clase **DbContext**, seguido de esto vamos a crear el constructor de la clase, en este caso vamos a crear el constructor de la clase y vamos a recibir como parámetro un objeto que instancia de la clase **DbContextOptions** en este caso el parámetro que recibe se llama **options** y va a ser de tipo **DbContextSistema**, con la sintaxis "**: base(options)**" le indicamos el parámetro al padre DbContext. Una vez realizado nuestro constructor vamos a crear un método llamado **OnModelCreating** que nos ayudará a mapear nuestras entidades con nuestro arreglo de datos y le estamos enviando como parámetro un objeto llamado **modelBuilder** que instancia a la clase **ModelBuilder**, dentro vamos a enviar al método OnModelCreating de la clase **base** ese modelBuilder como parámetro.

Abajo vamos a comenzar a mapear nuestras entidades indicándole como debe mapear las tablas de acuerdo a las entidades. Este mapeo se realizará en las clases anteriormente creadas alojadas dentro de la carpeta Mapping (ver figura 4.25)

```
public DbContextSistema(DbContextOptions<DbContextSistema> options) : base(options)
{

}
0 referencias | 0 excepciones
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    base.OnModelCreating(modelBuilder);
    modelBuilder.ApplyConfiguration(new VMClientMap());
    modelBuilder.ApplyConfiguration(new NetworkBondMap());
    modelBuilder.ApplyConfiguration(new OSFamilyMap());
    modelBuilder.ApplyConfiguration(new OSVersionMap());
    modelBuilder.ApplyConfiguration(new SQLFamilyMap());
    modelBuilder.ApplyConfiguration(new SQLVersionMap());
    modelBuilder.ApplyConfiguration(new VMTypeMap());
    modelBuilder.ApplyConfiguration(new PoolsMap());
    modelBuilder.ApplyConfiguration(new VPSMap());
    modelBuilder.ApplyConfiguration(new RolMap());
    modelBuilder.ApplyConfiguration(new UsuarioMap());
}
```

Figura 4.27 DbContextSistema.

A continuación un ejemplo del mapeamiento de la tabla **VMClient** representado en la clase **VMClientMap**.

VMClientMap.cs

Sistema.Datos — Sistema.Datos.Mapping.Clientes

```
1   using Microsoft.EntityFrameworkCore;
2   using Microsoft.EntityFrameworkCore.Metadata.Builders;
3   using Sistema.Entidades.Clientes;
4   using System;
5   using System.Collections.Generic;
6   using System.Text;
7
8   namespace Sistema.Datos.Mapping.Clientes
9   {
        1 referencia
10      public class VMClientMap : IEntityTypeConfiguration<VMClient>
11      {
            10 referencias | 0 excepciones
12          public void Configure(EntityTypeBuilder<VMClient> builder)
13          {
14              builder.ToTable("vmclient")
15                  .HasKey(c => c.idclient);
16          }
17      }
18  }
```

Figura 4.28 Clase VMClientMap.

En esta clase le vamos a indicar a la tabla su entidad correspondiente, en este caso la entidad VMClient se va a mapear con la tabla **vmclient** de la base de datos. Principalmente haremos nuestra clase pública y le vamos a indicar que herede de la clase **IEntityTypeConfiguration** y le vamos a enviar nuestra entidad, en este caso VMClient. Seguido de esto implementaremos nuestra interfaz donde utilizaremos el objeto **builder** que es una instancia de la clase **EntityTypeBuilder** de tipo VMClient que es la entidad. En este caso vamos a mapear el nombre de la tabla e igualmente la propiedad que su llave primaria.

VMClientMap.cs

Sistema.Datos — Sistema.Datos.Mapping.Clientes.VMClientMap

```
using Microsoft.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore.Metadata.Builders;
using Sistema.Entidades.Clientes;
using System;
using System.Collections.Generic;
using System.Text;

namespace Sistema.Datos.Mapping.Clientes
{
    1 referencia
    public class VMClientMap : IEntityTypeConfiguration<VMClient>
    {
        10 referencias | 0 excepciones
        public void Configure(EntityTypeBuilder<VMClient> builder)
        {
            builder.ToTable("vmclient")
                .HasKey(c => c.idclient);
        }
    }
}
```

Figura 4.29 Clase VMClientMap.

Volveremos a la clase contexto “DbContextSistema” y vamos a indicarle al modelBuilder que aplique la configuración con el comando **ApplyConfiguration** en este caso de VMClientMap (ver figura 4.27), ahora expondremos la colección de VMClient dentro del mismo DbContext, esto lo realizaremos con la siguiente línea de código (figura 4.30).

```
10 referencias | 0 excepciones
public DbSet<VMClient> VMClients { get; set; }
```

Figura 4.30 Exponer colección VMclient con el nombre VMClients.

4.6 Proyecto Web

4.6.1 Cadena de conexión

Como paso principal a trabajar en nuestro proyecto Web, vamos a agregar nuestro contexto de datos en el archivo **Startup.cs**, pero antes vamos a integrar la cadena de conexión en el archivo **appsettings.json**, a continuación veremos la sintaxis en la figura 4.31.

```
"ConnectionStrings": {
  "DefaultConnection": "Server=DELL-GOJIRA\\SQLEXPRESS;Database=proyectmsccf;User Id=sa; Password=Test1234"
},
```

Figura 4.31 Cadena de conexión.

Nuestra cadena de conexión se divide en tres parámetros, el nombre de nuestro servidor, el nombre de nuestra base de datos y el acceso al servidor, que en este caso es con el usuario de SQL Server.

Volvemos a nuestro archivo **Startup.cs** y en el método **ConfigureServices** vamos a agregar un servicio, en este caso a nuestra clase **services** vamos a agregar el contexto de datos de tipo DbContextSistema y en options le vamos a pasar la cadena de conexión utilizando el método **UseSqlServer** y le enviamos como parámetro de conexión la cadena de conexión a la base de datos llamada "**DefaultConnection**",

```
services.AddDbContext<DbContextSistema>(options =>
options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));
```

Figura 4.32 Integración del servicio de conexión.

Para no generar algún error agregaremos a dependencias las referencias a los proyectos Entidades y Datos de la solución.

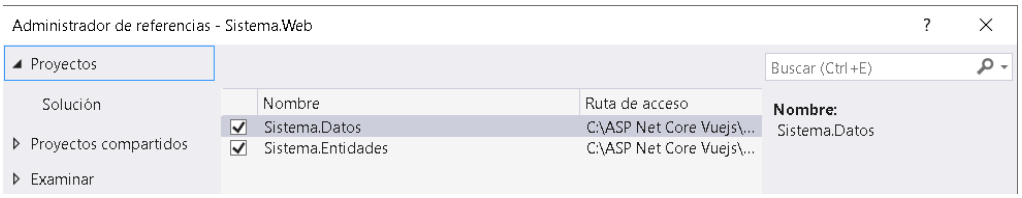

Figura 4.33 Administrador de referencias proyecto Web.

4.6.2 Controladores y modelos

Para agregar un controlador en nuestro proyecto Web, daremos clic derecho sobre la carpeta Controllers y seleccionaremos agregar/controlador (Figura 4.34).

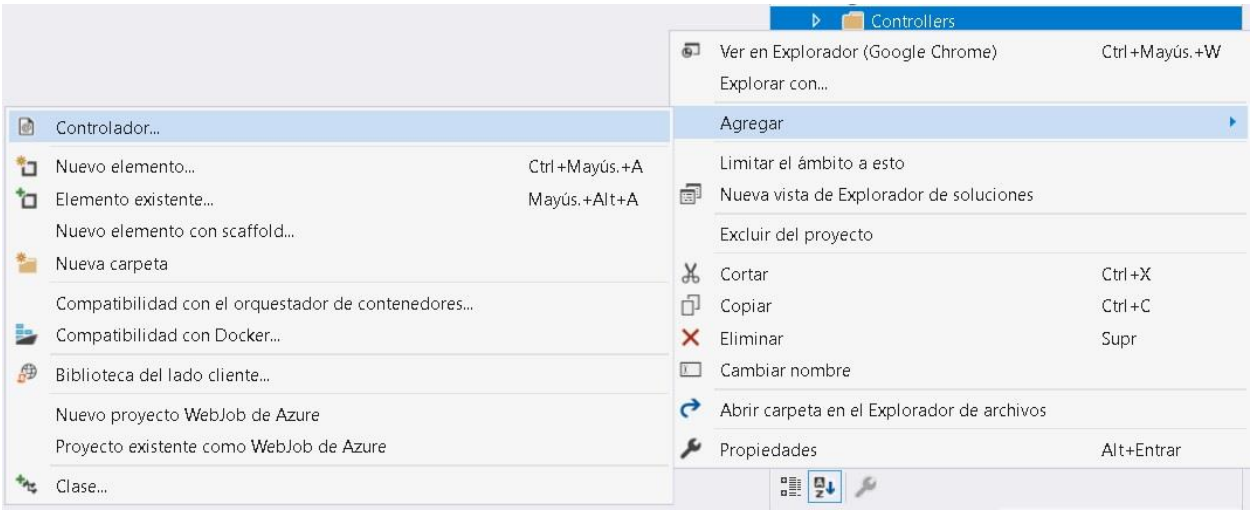

Figura 4.34 Agregar un Controlador.

Esta acción nos abrirá una ventana del asistente de Visual Studio en el cual seleccionaremos el tipo de controlador, en este caso seleccionaremos API Controller con acciones, usando Entity Framework (ver figura 4.35).

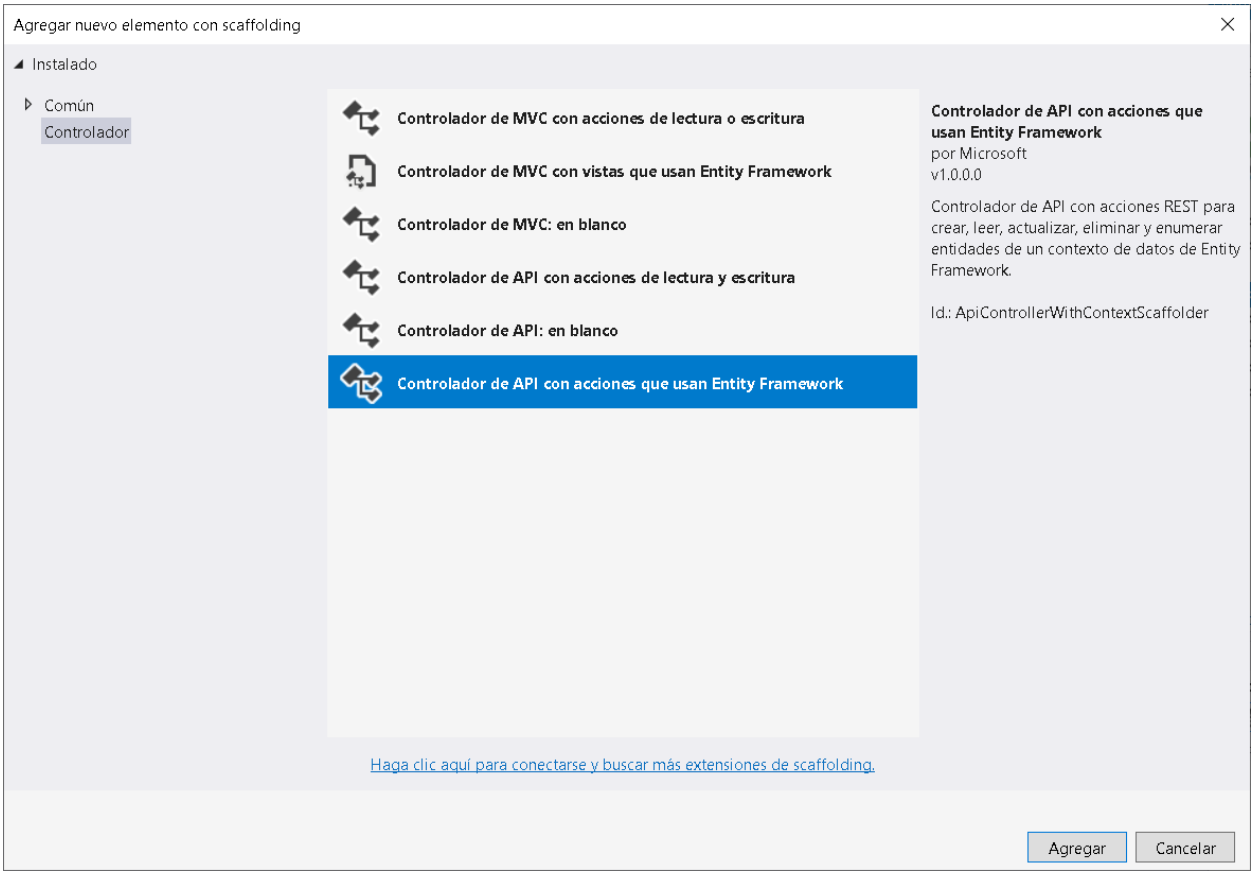

Figura 4.35 Asistente para crear controlador.

Una vez realizado nos aparece nuevamente el asistente donde seleccionaremos el modelo de datos y el contexto de datos, por defecto Visual Studio nos genera un nombre de controlador, nosotros podemos modificarlo a nuestro gusto o dejar el proporcionado, hecho esto daremos clic en agregar.

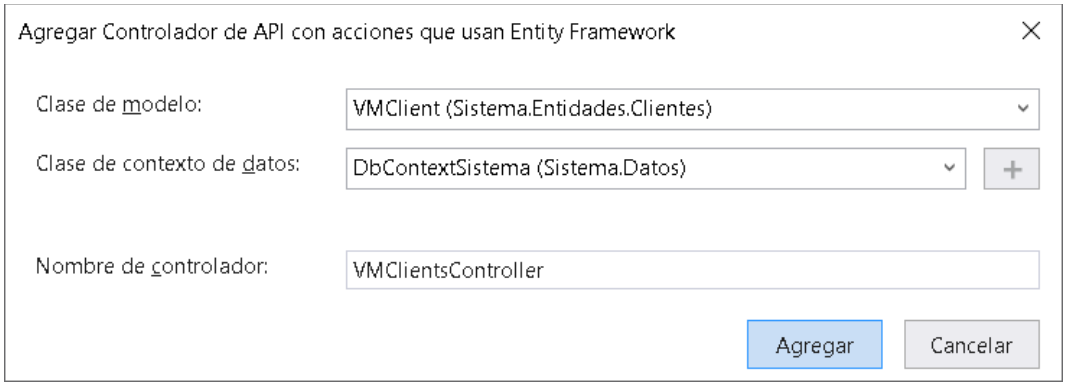

Figura 4.36 Agregar Controlador.

Hecho esto se ha generado nuestro controlador en la carpeta Controllers, por defecto se han creado los métodos con los llamados a los servicios RestFul, (Get, Put, Post y Delete), todos estos métodos van a ser modificados totalmente.

Antes de crear los métodos crearemos una carpeta en nuestro proyecto Web llamada Models, dentro de esta carpeta almacenaremos los modelos necesarios para cada uno de nuestros métodos, ya que no es correcto ya que así podremos reflejar lo que contiene la entidad, pero solo los datos que deseemos mostrar a los usuarios (figura 4.37).

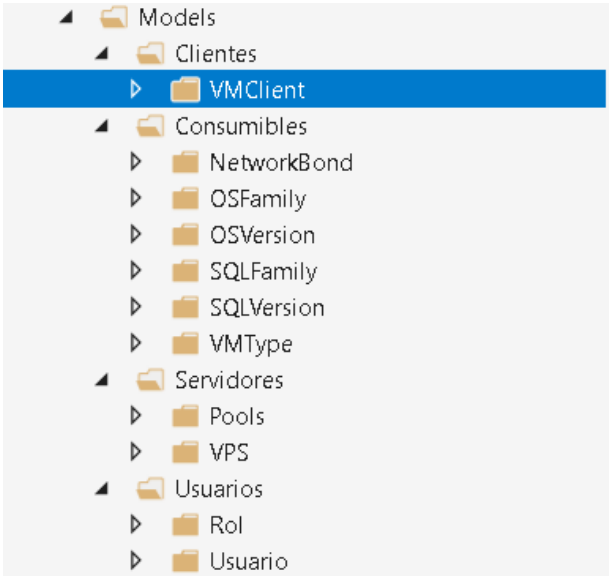

Figura 4.37 Carpeta Models del proyecto Web.

Tomaremos como ejemplo el modelo **UsuarioViewModel**, donde utilizando este modelo solo mostramos al usuario las propiedades que deseamos, en este caso no se incluye la propiedad "**password_salt**", a continuación veremos las propiedades del modelo Usuario y las propiedades del model UsuarioViewModel que será utilizado para la implementación de nuestro método Listar.

```
namespace Sistema.Entidades.Usuarios
{
    7 referencias
    public class Usuario
    {
        9 referencias | 0 excepciones
        public int idusuario { get; set; }
        [Required]
        3 referencias | 0 excepciones
        public int idrol { get; set; }
        [Required]
        [StringLength(50, MinimumLength = 3)]
        6 referencias | 0 excepciones
        public string nombre { get; set; }
        3 referencias | 0 excepciones
        public string direccion { get; set; }
        3 referencias | 0 excepciones
        public string telefono { get; set; }
        [Required]
        5 referencias | 0 excepciones
        public string email { get; set; }
        [Required]
        4 referencias | 0 excepciones
        public byte[] password_hash { get; set; }
        [Required]
        3 referencias | 0 excepciones
        public byte[] password_salt { get; set; }
        6 referencias | 0 excepciones
        public bool estado { get; set; }
        5 referencias | 0 excepciones
        public virtual Rol rol { get; set; }
        0 referencias | 0 excepciones
        public ICollection<VPS> vps { get; set; }
    }
}
```

Figura 4.38 Entidad Usuario.

```
namespace Sistema.Web.Models.Usuarios.Usuario
{
    2 referencias
    public class UsuarioViewModel
    {
        1 referencia | 0 excepciones
        public int idusuario { get; set; }
        1 referencia | 0 excepciones
        public int idrol { get; set; }
        1 referencia | 0 excepciones
        public string rol { get; set; }
        1 referencia | 0 excepciones
        public string nombre { get; set; }
        1 referencia | 0 excepciones
        public string direccion { get; set; }
        1 referencia | 0 excepciones
        public string telefono { get; set; }
        1 referencia | 0 excepciones
        public string email { get; set; }
        1 referencia | 0 excepciones
        public byte[] password_hash { get; set; }
        1 referencia | 0 excepciones
        public bool estado { get; set; }
    }
}
```

Figura 4.39 Modelo UsuarioViewModel.

Como podemos apreciar en nuestra entidad **Usuario** contiene más propiedades que el modelo **UsuarioViewModel**, además el modelo UsuarioViewModel no contiene la relación que existe entre las tablas. En resumen, los modelos que se encuentran dentro del proyecto Web son una forma de restringir a la aplicación y al usuario las propiedades que puede visualizar de cada entidad en las vistas de la herramienta.

Se crearon modelos para generar el listado de los datos, actualizar, crear, y seleccionar. Estos modelos serán implementados en los métodos de la aplicación.

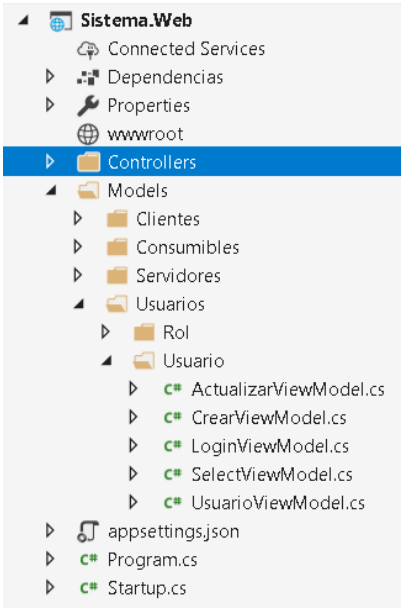

Figura 4.40 Modelos del controlador Usuario.

Como podemos observar en la figura 4.40 se crearon 5 modelos para el controlador **UsuariosController**.

4.6.3 Métodos CRUD

Los métodos CRUD nos permiten realizar operaciones básicas dentro de cada una de nuestras entidades, las cuales son Agregar, Consultar, Modificar y Eliminar (por sus siglas en ingles CRUD Create, Read, Update and Delete), cada una de estas operaciones pueden ser utilizadas en las entidades que deseemos, en este proyecto serán implementadas en la mayoría a excepción de la entidad Roles, esta entidad solo tendrá acceso al método consultar.

- **Método Listar**

Para la explicación de estos métodos nos apoyaremos del controlador VMClientsController como ejemplo para su interpretación. Como primer método a analizar será el de consultar en este caso llamado Listar, ya que la acción principal que realizamos con una entidad es verificar los datos que contiene, o de otra forma seria el listar cada uno de los registros.

```
// GET: api/VMClients/Listar
[HttpGet("[action]")]
0 referencias | 0 solicitudes | 0 excepciones
public async Task <IEnumerable<VMClientViewModel>> Listar()
{
    var vmclient = await _context.VMClients.ToListAsync();
    return vmclient.Select(c => new VMClientViewModel
    {
        idclient = c.idclient,
        clientname = c.clientname,
        clientfullname = c.clientfullname,
        clientemail = c.clientemail,
        clientphone = c.clientphone,
        clientcontact = c.clientcontact,
        emailcontact_tecnico = c.emailcontact_tecnico,
        estado = c.estado
    });
}
```

Figura 4.41 Método Listar del controlador VMClientsController

Para acceder a este método la ruta será como se muestra en la figura 4.41 **api/VMClients/Listar**, como podemos observar este será un método que será una tarea asíncrona donde utilizaremos el modelo **VMClientViewModel** de nombre **Listar**, esta tarea devolverá un

objeto de tipo **IEnumerable** que tiene la estructura del modelo VMClientViewModel. Dentro de este método listar declaramos un objeto llamado **vmclient** para obtener el registro de clientes de maquitas virtuales utilizando la colección expuesta en el contexto, la colección expuesta en el contexto es llamada **VMClients**, vamos a obtener la lista del registro con el método **ToListAsync**. De esta forma ya tenemos la lista en un objeto llamado vmclient. Procediendo con nuestro método ahora retornaremos los datos de acuerdo a la estructura del modelo utilizado anteriormente y al utilizar este modelo mostraremos al usuario solo los datos que nosotros deseamos, como podemos observar no nostramos el Id de nuestro cliente, solo los datos que consideramos fundamentales para su exportación.

- **Método Mostrar**

Una vez creado el método listar, que es el resultado de exportar todos los registros de una entidad procederemos a nuestro método Mostrar, el cual nos permite seleccionar uno de estos registros en específico. Para obtener una mayor comprensión veremos la figura 4.42.

```
// GET: api/VMClients/Mostrar/5
[HttpGet("[action]/{id}")]
0 referencias | 0 solicitudes | 0 excepciones
public async Task<IActionResult> Mostrar([FromRoute] int id)
{
    var vmclient = await _context.VMClients.FindAsync(id);

    if (vmclient == null)
    {
        return NotFound();
    }
    return Ok(new VMClientViewModel{
        idclient = vmclient.idclient,
        clientname = vmclient.clientname,
        clientfullname = vmclient.clientfullname,
        clientemail = vmclient.clientemail,
        clientphone = vmclient.clientphone,
        clientcontact = vmclient.clientcontact,
        emailcontact_tecnico = vmclient.emailcontact_tecnico,
        estado = vmclient.estado
    });
}
```

Figura 4.42 Método Mostrar del controlador VMClientsController.

Como podemos observar para acceder a la ruta de este método agregaremos una propiedad, la cual será el **Id** del cliente de máquina virtual, como ejemplo la ruta seria **api/VMClients/Mostrar/1** y el resultado de esta liga será mostrar todas las propiedades que deseemos del cliente que cuente con el Id 1. Este método será de igual forma de tipo público y será una tarea asíncrona que va a retornar un objeto del tipo **IActionResult** que espera como parámetro un valor llamado **id**, este parámetro deberá ser proporcionado en la ruta como se mencionó anteriormente. Dentro del método mostrar declaramos un objeto llamado **vmclien**t que va a obtener el registro cuyo valor del campo **idvmclient** coincide con el parámetro **id** que estamos recibiendo. Para realizar la búsqueda de registro que coincide con el id utilizamos el método **FindAsync**. Si el registro no existe retornamos un **NoTFound** indicando que no ha sido encontrado el registro con el valor proporcionado. Si el registro ha sido encontrado lo retornaremos utilizando la estructura del modelo **VMClientViewModel**.

- **Método Actualizar**

Una vez tenemos nuestro método mostrar que nos detalla la información completa de un registro de una entidad del diccionario de datos, otra de las actividades que solemos hacer es la de modificar o actualizar la información de este, para eso utilizaremos el siguiente método llamado **Actualizar**. En este caso el método será un **HttpPut** que llamará a la acción Actualizar en el cual no le enviaremos solo el id, le enviaremos el objeto completo (registro), para este método utilizaremos un nuevo modelo llamado **ActualizarViewModel**, una vez recibido el objeto, el método valida que la información nueva esté dentro de los parámetros acotados en las **DataAnnotations** del modelo ActualizarViewModel. Si los nuevos valores cumplen con los requerimientos ya implementados dentro del modelo se procede a validar el **id** a actualizar. Validados los datos se creará una variable llamada **vmclient** donde consultaremos que el objeto exista con el método

FirstOrDefaultAsync que devuelve el primer registro que encuentra utilizando el parámetro **idclient** y lo asigna a la propiedad **idclient** del modelo que estamos recibiendo (ver figura 4.43).

```
// PUT: api/VMClients/Actualizar
[HttpPut("[action]")]
0 referencias | 0 solicitudes | 0 excepciones
public async Task<IActionResult> Actualizar([FromBody] ActualizarViewModel model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    if (model.idclient <= 0)
    {
        return BadRequest();
    }
    var vmclient = await _context.VMClients.FirstOrDefaultAsync(c => c.idclient == model.idclient);

    if (vmclient == null)
    {
        return NotFound();
    }
    vmclient.clientname = model.clientname;
    vmclient.clientfullname = model.clientfullname;
    vmclient.clientemail = model.clientemail;
    vmclient.clientphone = model.clientphone;
    vmclient.clientcontact = model.clientcontact;
    vmclient.emailcontact_tecnico = model.emailcontact_tecnico;
    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        //Guardar Excepción
        return BadRequest();
    }
    return Ok();
}
```

Figura 4.43 Método Actualizar del controlador VMClientsController.

Una vez hecho esto validamos que el objeto vmclient no se encuentre vacío, de no ser así se asignan los valores de las propiedades del modelo ActualizarViewModel a las propiedades del registro seleccionado. Después creamos un **try catch** en donde si todas las validaciones son correctas se guardarán los cambios, de lo contrario se mandará a consola un mensaje de error de la herramienta.

- **Método Crear**

Para nuestro método **Crear** la ruta será **api/VMClients/Crear**, para este método utilizaremos un modelo de datos llamado **CrearViewModel** en donde podemos utilizar **DataAnnotations** para la validación de nuestros datos. En este caso para este modelo no utilizamos la propiedad **Id** de la entidad, ya que esta se genera de forma automática cada nuevo registro (ver figura 4.21).

```
// POST: api/VMClients/Crear
[HttpPost("[action]")]
0 referencias | 0 solicitudes | 0 excepciones
public async Task<IActionResult> Crear([FromBody] CrearViewModel model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    VMClient vMClient = new VMClient
    {
        clientname = model.clientname,
        clientfullname = model.clientfullname,
        clientemail = model.clientemail,
        clientphone = model.clientphone,
        clientcontact = model.clientcontact,
        emailcontact_tecnico = model.emailcontact_tecnico,
        estado = true
    };
    _context.VMClients.Add(vMClient);
    try
    {
        await _context.SaveChangesAsync();
    }
    catch(Exception ex)
    {
        return BadRequest();
    }
    return Ok();
}
```

Figura 4.44 Método Crear del controlador VMClientsController.

Para nuestro método crear vamos a desarrollar un objeto llamado **model**, que será una instancia de la clase **CrearViewModel**, como primera validación tenemos la de **ModelState** que verificará que se cumplan las validaciones establecidas dentro de nuestro modelo CrearViewModel. Después utilizando la entidad **VMClient** vamos a crear un objeto llamado **vMClient**, hecho esto asignaremos los valores del objeto **model** al objeto vMClient para proceder, realizado esto le

asignaremos el objeto vMClient al contexto de datos llamado **VMClients** para proceder a guardar los cambios dentro de la entidad utilizando un **try catch** donde si hubiese un error retornaremos un **BadRequest**.

- **Método Desactivar/Activar**

Dentro de los métodos CRUD existe el de **Eliminar (Delete)** que es un procedimiento para eliminar un registro dentro de una tabla de base de datos, eliminar un registro dentro de una herramienta para una organización no es un procedimiento bien visto, para solucionar esto implementamos una propiedad dentro de las entidades llamada **estado** que es de tipo **boolean**, donde sus valores posibles es 1 o 0 para nuestro gestor de base de datos y para nuestro IDE los valores serán verdadero y falso (true and false), dentro de nuestras vistas visualizaremos el campo como Activo y Suspendido para una mejor comprensión del usuario (figura 4.45).

```
// PUT: api/VMClients/Desactivar/1
[HttpPut("[action]/{id}")]
0 referencias | 0 solicitudes | 0 excepciones
public async Task<IActionResult> Desactivar([FromRoute] int id)
{
    if (id <= 0)
    {
        return BadRequest();
    }
    var vmclient = await _context.VMClients.FirstOrDefaultAsync(c => c.idclient == id);
    if (vmclient == null)
    {
        return NotFound();
    }
    vmclient.estado = false;
    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        //Guardar Excepción
        return BadRequest();
    }
    return Ok();
}
```

Figura 4.45 Método Desactivar del controlador VMClientsController

Este método **Desactiva**r es similar al método **Actualizar**, la diferencia es que no recibimos el objeto completo si no que el dato requerido es el **Id**, quedando como ruta para nuestro método de la siguiente manera **api/VMClients/Desactivar/{id}**, como primer paso dentro de nuestro método es hacer una validación que nos permita verificar que el Id de queremos desactivar sea correcto. Hecho esto utilizando el método **FirstOrDefaultAsync** validaremos que el Id exista. Si el Id no existiera retornaríamos un **NotFound**, notificando al usuario que no fue posible desactivar el Id. Si todo es correcto modificaremos el valor de la propiedad a falso **(false)**, utilizaremos un **try catch** para guardar los datos o notificarle al usuario un **BadRequest** de que no fue posible hacer el cambio.

Para nuestro método **Activar** es casi el mismo procedimiento, la diferencia es que asigna el valor de verdadero **(true)** a la propiedad **estado** (ver figura 4.46).

```
// PUT: api/VMClients/Activar/1
[HttpPut("[action]/{id}")]
0 referencias | 0 solicitudes | 0 excepciones
public async Task<IActionResult> Activar([FromRoute] int id)
{
    if (id <= 0)
    {
        return BadRequest();
    }
    var vmclient = await _context.VMClients.FirstOrDefaultAsync(c => c.idclient == id);
    if (vmclient == null)
    {
        return NotFound();
    }
    vmclient.estado = true;
    try
    {
        await _context.SaveChangesAsync();
    }
    catch (DbUpdateConcurrencyException)
    {
        //Guardar Excepción
        return BadRequest();
    }
    return Ok();
}
```

Figura 4.46 Método Activar del controlador VMClientsController.

4.7 Vistas Vuetify

Una vez creados todos los métodos necesarios para el funcionamiento de nuestra herramienta web, procederemos a crear las vistas de cada una de las entidades donde se llevará acabo su función. Para el desarrollo de las vistas de nuestra herramienta, también llamado front-end utilizaremos el Framework Vuetify como se indicó en el capítulo 2.

4.7.1 Creando el proyecto con VueJS CLI

Para la creación de nuestro proyecto en VueJS necesitamos instalar NodeJS en nuestro Visual Studio Code, para descargar este entorno de ejecución para JavaScript debemos acceder a su sitio oficial http://nodejs.org/es. Una vez accedemos a la siguiente dirección web nos aparecerá la siguiente ventana (ver figura 4.47).

Figura 4.47 Descarga NodeJS.

Procederemos a descargar e instalar la versión 12.16.1 que es la versión más estable. La instalación del complemento para Visual Studio Code es muy sencilla e intuitiva. Ya instalado Nodejs procederemos a instalar Vue cli, para instalarlo accederemos a nuestra terminal cmd de Windows y escribiremos el siguiente comando (ver figura 4.48).

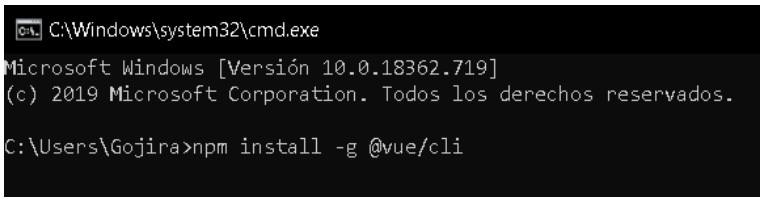

Figura 4.48 Instalar Vue/cli.

Una vez instalado procederemos a crear el proyecto en Vue/cli, como primer caso crearemos un directorio en el cual almacenaremos todos los archivos que se nos generen. Hecho esto accederemos a nuestro directorio desde Visual Studio Code y abriremos su terminal. Para crear nuestro proyecto escribiremos el siguiente código (ver figura 4.49).

Figura 4.49 Código crear proyecto Vue

Hecho esto indicaremos que paquetes vamos a utilizar en nuestro proyecto, hecho esto finalizaremos la creación de nuestro proyecto Vue, si observamos nuestro directorio anteriormente creado se habrán generado nuevos directorios que incluyen las vistas y paquetes necesarios para el desarrollo de nuestro front-end (figura 4.50).

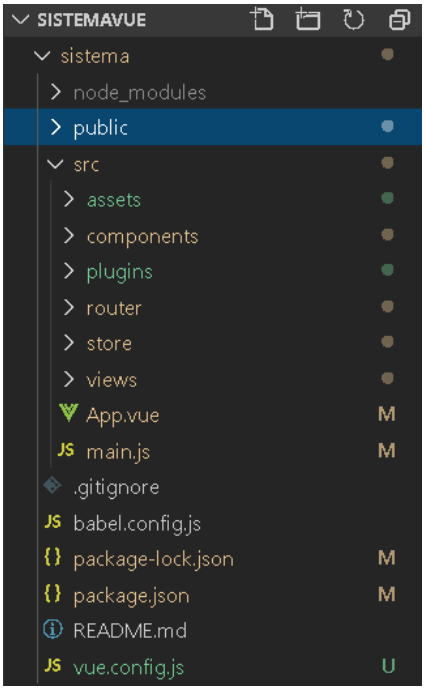

Figura 4.50 Directorio proyecto Vue.

4.7.2 Añadir Vuetify al proyecto Vue

Para agregar **Vuetify** a nuestro proyecto accederemos a su directorio desde **Visual Studio Code**, y desde nuestra terminal ejecutaremos el siguiente código: (ver figura 4.51).

Figura 4.51 Código para añadir Vuetify al proyecto Vue.

Al realizar esta acción antes de finalizar la terminal nos preguntará si deseamos añadir una plantilla personalizada a lo cual elegiremos si, si utilizaremos **CSS** a lo cual afirmaremos. Una vez finalizado en nuestro componente **App.vue** configuraremos nuestra plantilla principal para nuestra SPA.

4.7.3 Creación de vistas

El desarrollo de una vista está dividido en 3 etapas fundamentales, el diseño y creación de métodos **JavaScript** para su funcionamiento, agregar la vista al listado de rutas y agregarla a nuestro menú de la página SPA.

Consideramos el diseño y creación de métodos JavaScript como una sola etapa porque ambos se estructuran en el mismo componente **VueJS**. El diseño de un componente Vuejs se refiere a toda la codificación **HTML** necesaria para proporcionar un entorno agradable para el usuario. Los métodos **js** los utilizamos para tener una interacción con el entorno, permitiéndonos trabajar de forma simultánea con los métodos creados en nuestros controladores del proyecto web utilizando **Axios** para su llamado.

Como ejemplo utilizaremos la vista **VMClient.vue**, lo que se expone en la figura 4.52 es la organización de código HTML para la creación de un modal que utilizaremos para crear un nuevo registro y actualizar un registro ya existente.

```
<v-dialog v-model="dialog" max-width="500px">
    <template v-slot:activator="{ on }">
    <v-btn color="primary" dark class="mb-2" v-on="on">Nuevo</v-btn>
    </template>
        <v-container>
        <v-row>
            <v-layout wrap>
            <v-flex xs12 sm12 md12>
            <v-text-field v-model="clientname" label="Cliente"></v-text-field>
            </v-flex>
            <v-flex xs12 sm12 md12>
            <v-text-field v-model="clientfullname" label="Nombre Completo"></v-text-field>
            </v-flex>
            <v-flex xs12 sm12 md12>
            <v-text-field v-model="clientemail" label="Email"></v-text-field>
            </v-flex>
            <v-flex xs12 sm12 md12>
            <v-text-field v-model="clientphone" label="Teléfono"></v-text-field>
            </v-flex>
            <v-flex xs12 sm12 md12>
            <v-text-field v-model="clientcontact" label="Contacto Técnico"></v-text-field>
            </v-flex>
            <v-flex xs12 sm12 md12>
            <v-text-field v-model="emailcontact_tecnico" label="Email Contacto Técnico"></v-text-field>
            </v-flex>
            <v-flex xs12 sm12 md12 v-show="valida">
            <div class="red--text" v-for="v in validaMensaje" :key="v" v-text="v">
            </div>
            </v-flex>
            </v-layout>
        </v-row>
        </v-container>
    </v-card-text>
```

Figura 4.52 Código Modal del componente VMClient.vue.

Con esta codificación le indicamos a nuestra herramienta web como debe mostrar al usuario el llenar un nuevo registro y al mismo tiempo indicamos a la aplicación en que variables guardar esos valores para después ser utilizados por un método js. Este modal como se mencionó anteriormente puede ser utilizado para guardar un nuevo registro y actualizar uno existente. Para lograr esto creamos un método llamado **guardar** el cual hace el llamado a dos métodos del controlador **VMClientsController** mediante axios.

```
guardar () {
 if (this.validar()){
   return;
 }
 let header={"Authorization" : "Bearer " + this.$store.state.token};
 let configuracion= {headers : header}
 if (this.editedIndex > -1) {
     //Codigo para editar
     let me = this;
     axios.put('api/VMClients/Actualizar', {
       'idclient': me.id,
       'clientname': me.clientname,
       'clientfullname' : me.clientfullname,
       'clientemail' : me.clientemail,
       'clientphone': me.clientphone,
       'clientcontact': me.clientcontact,
       'emailcontact_tecnico': me.emailcontact_tecnico
     },configuracion).then(function(response){
       me.close();
       me.listar();
       me.limpiar();
     }).catch(function(error){
       console.log(error);
     });
 } else {
     //Codigo para guardar
     let me = this;
     axios.post('api/VMClients/Crear', {
       'clientname': me.clientname,
       'clientfullname' : me.clientfullname,
       'clientemail' : me.clientemail,
       'clientphone': me.clientphone,
       'clientcontact': me.clientcontact,
       'emailcontact_tecnico': me.emailcontact_tecnico
     },configuracion).then(function(response){
       me.close();
       me.listar();
       me.limpiar();
     }).catch(function(error){
       console.log(error);
     });
 }
},
```

Figura 4.53 Método guardar del componente VMClient.vue

Podemos observar en la figura 4.53 el llamado al método **Crear** y **Actualizar** utilizando las librerías de **axios**, de igual forma podemos apreciar que seguido de la ejecución de estos métodos mandamos a llamar los métodos **cerrar**, que cerrara la ventana modal que abrimos al presionar el botón nuevo o actualizar, el método **listar**, que nos genera un listado de los registros de la entidad **VMClient** para poder visualizar los cambios que acabamos de realizar y el método **limpiar**, que limpiará cada una de las cajas de texto de nuestro modal y así podamos guardar o actualizar un nuevo registro.

Una vez creada nuestra vista debemos indicarle a nuestra herramienta que hemos creado un nuevo componente, así que la agregaremos a nuestro registro de rutas que se encuentra dentro del directorio Router dentro del directorio de nuestra solución para poder hacer uso de ella (ver figura 4.54).

```
{
  path: '/vmclients',
  name: 'vmclients',
  component: VMClient,
  meta:{
    administrador:true,
    soporte:true,
    gerente:true
  }
},
```

Figura 4.54 Enrutamiento componente VMClient.

Hecho esto podemos hacer uso de nuestra entidad, le hemos creado una dirección de acceso a nuestro componente, realizado esto agregaremos el enlace a nuestro menú de la herramienta web. Esto lo lograremos utilizando el código de la figura 4.55.

```
<v-list-item :to="{ name: 'vmclients'}">
  <v-list-item-action>
    <v-icon>table_chart</v-icon>
  </v-list-item-action>
  <v-list-item-content>
    <v-list-item-title>
      VMClients
    </v-list-item-title>
  </v-list-item-content>
</v-list-item>
```

Figura 4.55 Agregar vista al menú de la herramienta.

4.8 Gestión de usuarios

Dentro de la herramienta web existen 3 roles disponibles para proporcionar un control de acceso a la misma, los roles existentes son **Administrador** que hace referencia al encargado del área de sistemas computacionales, **Soporte** que es el rol asignado a los empleados que proporcionan soporte técnico dentro de la institución y el rol **Gerente**, que es el encargado de realizar las notificaciones a los clientes referente a los pagos.

Actualmente la gestión de la entidad **Usuarios** controlada por el controlador **UsuariosController** está asignada al **Rol** Administrador en nuestro **back-end**, como se muestra en la figura 4.56 podremos ver el método **Crear**, al cual solo podrá tener acceso un usuario que tenga el rol de Administrador, para establecer que rol está autorizado utilizamos el método **Authorize** que recibe como parámetro la variable **Roles**, de esta forma nosotros brindamos a la herramienta una forma de gestionar todos los métodos de los controladores. Podemos permitir el acceso a varios roles utilizando el método Authorize, para este proyecto toda la gestión de usuarios será llevada por el rol Administrador.

```
// POST: api/Usuarios/Crear
[Authorize(Roles = "Administrador")]
[HttpPost("[action]")]
0 referencias | 0 solicitudes | 0 excepciones
public async Task<IActionResult> Crear([FromBody] CrearViewModel model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    var email = model.email.ToLower();
    if (await _context.Usuarios.AnyAsync(u => u.email == email))
    {
        return BadRequest("El email ya existe.");
    }
    CrearPasswordHash(model.password, out byte[] passwordHash, out byte[] passwordSalt);
    Usuario usuario = new Usuario
    {
        idrol = model.idrol,
        nombre = model.nombre,
        direccion = model.direccion,
        telefono = model.telefono,
        email = model.email.ToLower(),
        password_hash = passwordHash,
        password_salt = passwordSalt,
        estado = true
    };
    _context.Usuarios.Add(usuario);
    try
    {
        await _context.SaveChangesAsync();
    }
    catch(Exception ex)
    {
        return BadRequest();
    }
    return Ok();
}
```

Figura 4.56 Método Crear autorizado al rol Administrador.

De esta forma tenemos un control para la correcta autorización a la gestión de usuarios en nuestro back-end. De igual manera debemos proporcionar un reflejo de esta lógica en nuestro **front-end** para que al momento de acceder con un usuario con el rol de administrador nos permita visualizar esta vista, de acceder con un usuario con un rol diferente no nos muestre la opción en el menú y tampoco nos permita el acceso ingresando la **url** pertinente al navegador.

```
<template v-if="esAdministrador">
  <v-list-group>
    <v-list-item slot="activator">
      <v-list-item-content>
        <v-list-item-title>
          Accesos
        </v-list-item-title>
      </v-list-item-content>
    </v-list-item>
    <v-list-item :to="{ name: 'rols'}">
      <v-list-item-action>
        <v-icon>table_chart</v-icon>
      </v-list-item-action>
      <v-list-item-content>
        <v-list-item-title>
          Roles
        </v-list-item-title>
      </v-list-item-content>
    </v-list-item>
    <v-list-item :to="{ name: 'usuarios'}">
      <v-list-item-action>
        <v-icon>table_chart</v-icon>
      </v-list-item-action>
      <v-list-item-content>
        <v-list-item-title>
          Usuarios
        </v-list-item-title>
      </v-list-item-content>
    </v-list-item>
  </v-list-group>
</template>
```

Figura 4.57 Menú Acceso de la página SPA.

Como podemos observar en la figura 4.57 en menú Accesos solo puede ser visualizado por un usuario que tenga el rol Administrador, este menú nos da acceso a las entidades Roles y a la entidad Usuarios.

4.9 Servicio Notificaciones vía correo electrónico

Para notificar vía correo electrónico el primer paso que necesitamos desarrollar es la creación de un servicio de correo que sea capaz de comunicarse con un servidor de correo electrónico, en esta ocasión el gestor de correo que utilizaremos será Gmail. Para utilizar este gestor de forma correcta activaremos la opción Acceso a aplicaciones menos seguras, por defecto esta opción se encuentra deshabilitada, lo que haremos será habilitarla como se muestra en la figura 4.58.

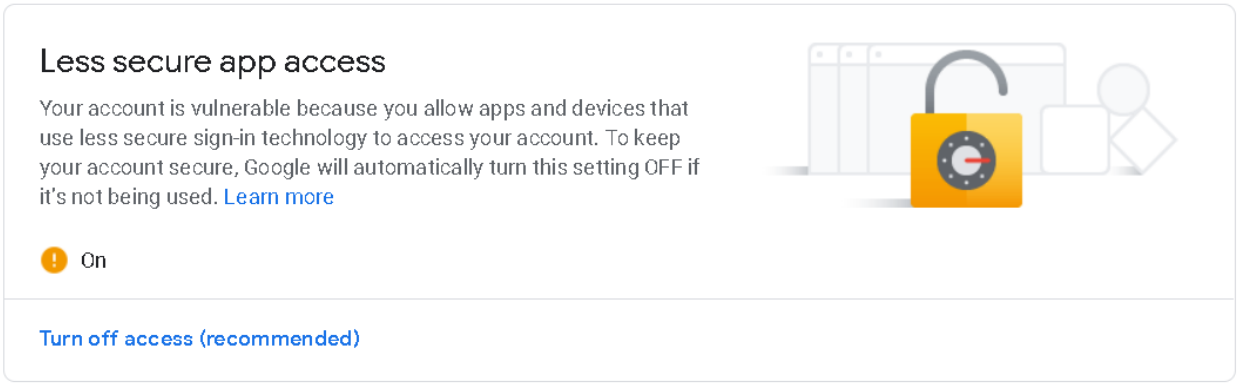

Figura 4.58 Acceso a aplicaciones menos seguras.

Realizado esto podemos proseguir con nuestra función de correo en nuestra aplicación, utilizaremos una clase nueva y la llamaremos **Correo** que se encontrara dentro del directorio **Funciones**, en esta función utilizaremos la clase **smtp**, que se encarga de realizar la conexión con el servidor de correo y al mismo tiempo se configura la seguridad para la misma (ver figura 4.59).

```
public Correo()
{
    smtp.Host = "smtp.gmail.com";
    smtp.Port = 587;
    smtp.UseDefaultCredentials = false;
    smtp.Credentials = new System.Net.NetworkCredential("angrdzrayo@gmail.com", "Password");
    smtp.EnableSsl = true;
}
```

Figura 4.59 Método Correo.

Una vez creado este método procederemos a crear el método principal para el envío de correo electrónico que será llamado **ConfigurarMail**, que recibirá un listado de los correos hábiles para el envío de notificaciones, en este método haremos uso de la clase **MailMessage**, la cual usaremos para darle forma a nuestro correo electrónico, esto consiste en elegir los destinatarios, agregar un asunto y contenido o cuerpo del mensaje (ver figura 4.60).

```
public static void ConfigurarMail(List<string> Receptores, int op)
{
    try
    {
        Correo ec = new Correo();
        MailMessage mnj = new MailMessage();
        foreach (var correo in Receptores)
        //mnj.To.Add(new MailAddress(correo));
        mnj.Bcc.Add(new MailAddress(correo));
        mnj.From = new MailAddress("angrdzrayo@gmail.com", "BTU Cloud");
        switch (op)
        {
            case 1:
                ...
            case 2:
                ...
        }
        mnj.IsBodyHtml = true;
        mnj.Priority = MailPriority.Normal;
        ec.EnviarMensaje(mnj);
    }
    catch (Exception e)
    {
        Console.WriteLine(e);
    }
}
```

Figura 4.60 Método ConfigurarMail.

Teniendo el cuerpo del mensaje listo se procede a crear el último método que será el encargado de enviar el mensaje apoyado de la clase smtp, el método es el siguiente (ver figura 4.61).

```
public void EnviarMensaje(MailMessage mensaje)
{
    smtp.Send(mensaje);
}
```

Figura 4.61 Método EnviarMensaje.

En conclusión, tenemos un método encargado de realizar la conexión con nuestro gestor de correo, uno encargado de recopilar los datos necesarios para armar el cuerpo del mensaje y por último uno encargado de enviar el mensaje.

4.9.1 Automatización de envío de notificaciones vía correo electrónico

Para automatizar el envío de notificaciones es necesario crear un servicio que se esté ejecutando en segundo plano en todo momento, para lograr esto necesitamos cumplir un cierto número de condiciones, esto es el análisis de las fechas en las cuales se enviará la notificación vía correo electrónico a los usuarios que cuenten con un servicio de servidor virtual activo, y solo si su estatus de notificación se encuentre activo, esta opción podrá ser deshabilitada por el usuario si el cliente ya ha realizado el pago.

```
public Task StartAsync(CancellationToken cancellationToken)
{
    //Inicia el evento FromSeconds(60));
    timer = new Timer(DoWork, null, TimeSpan.Zero, TimeSpan.FromDays(1));
    return Task.CompletedTask;

}
1 referencia | 0 excepciones
private void DoWork(object state)
 {
    var fechaActual = DateTime.Now;
    var A = fechaActual.Year;
    var M = fechaActual.Month;
    //Realiza la opción principal
    DateTime fechaEs = new DateTime(A,M,10);
    DateTime fechaEs2 = new DateTime(A, M, 12);
    var resul = fechaActual.CompareTo(fechaEs);
    if (fechaActual.ToShortDateString() == fechaEs.ToShortDateString())
    {
        using (IServiceScope scope = _provider.CreateScope())
        {
            var context = scope.ServiceProvider.GetRequiredService<DbContextSistema>();
            var CorreosNotifi = context.Notifivpss.Where(x => x.estado == true).Select(x => x).Where(xt => xt.idnotivps > 0).Select(x => x.emailcontact_tecnico).ToList();
            Correo.ConfigurarMail(CorreosNotifi, 1);
        }
    }
    else if (fechaActual.ToShortDateString() == fechaEs2.ToShortDateString())
    {
        using (IServiceScope scope = _provider.CreateScope())
        {
            var context = scope.ServiceProvider.GetRequiredService<DbContextSistema>();
            var CorreosNotifi = context.Notifivpss.Where(x => x.estado == true).Select(x => x).Where(xt => xt.idnotivps > 0).Select(x => x.emailcontact_tecnico).ToList();
            Correo.ConfigurarMail(CorreosNotifi, 2);
        }
    }
    else
    {
        Console.WriteLine("No paso nada");
    }
}
```

Figura 4.62 Servicio en segundo plano.

Como podemos observar en el método principal se realiza una comparación con la fecha actual y la fecha predefinida para la notificación, si estas fechas coinciden se procede a realizar un conjunto de tareas, que son la de utilizar el servicio **Scope** para tener acceso a nuestro **contexto de datos**, seguido de esto creamos un listado de los correos válidos para ser notificados, una vez que hemos obtenido esta información será utilizada por la clase correo en sincronía del método

ConfigurarMail, el cual en su constructor recibe un listado de cadenadas de texto como se muestra en la figura 4.60.

Capítulo 5 Resultados y Conclusiones

En este capítulo se presentan los resultados tras haber realizado el desarrollo de la herramienta **SYSVPS**, se mostrarán cada uno de los módulos que con lleva la herramienta comenzando por el inicio de sesión del usuario.

Se analizarán los resultados obtenidos y se realizan las conclusiones pertinentes de la herramienta y los posibles trabajos a futuro que se pueden realizar utilizando esta herramienta como base.

5.1 Inicio de sesión

Como pantalla principal de nuestra herramienta tendremos el Inicio de sesión o acceso al sistema, donde los datos requeridos para realizar esta acción son el correo electrónico del usuario y su contraseña (ver figura 5.1).

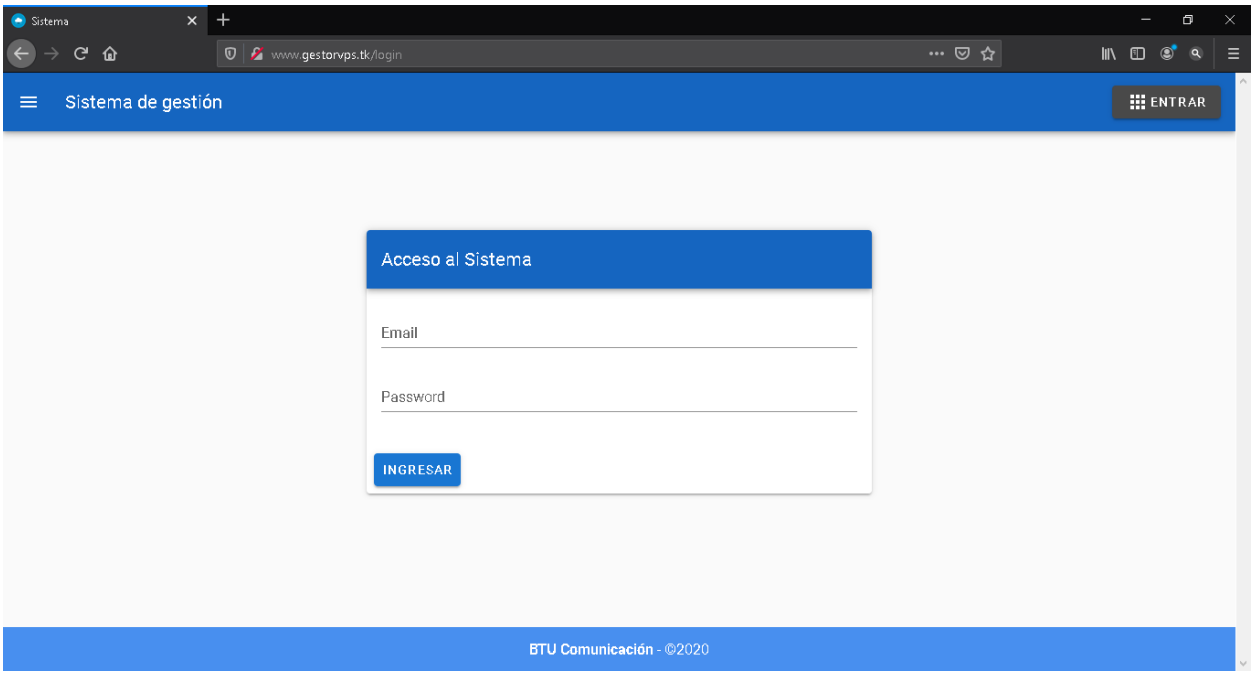

Figura 5.1 Acceso al sistema.

Si nosotros nos equivocásemos al tratar de ingresar a nuestro sistema introduciendo un email no registrado o una contraseña errónea se nos mostrará una advertencia de error ilustrada en color rojo indicándonos que uno de los datos es incorrecto como se muestra en la figura 5.2.

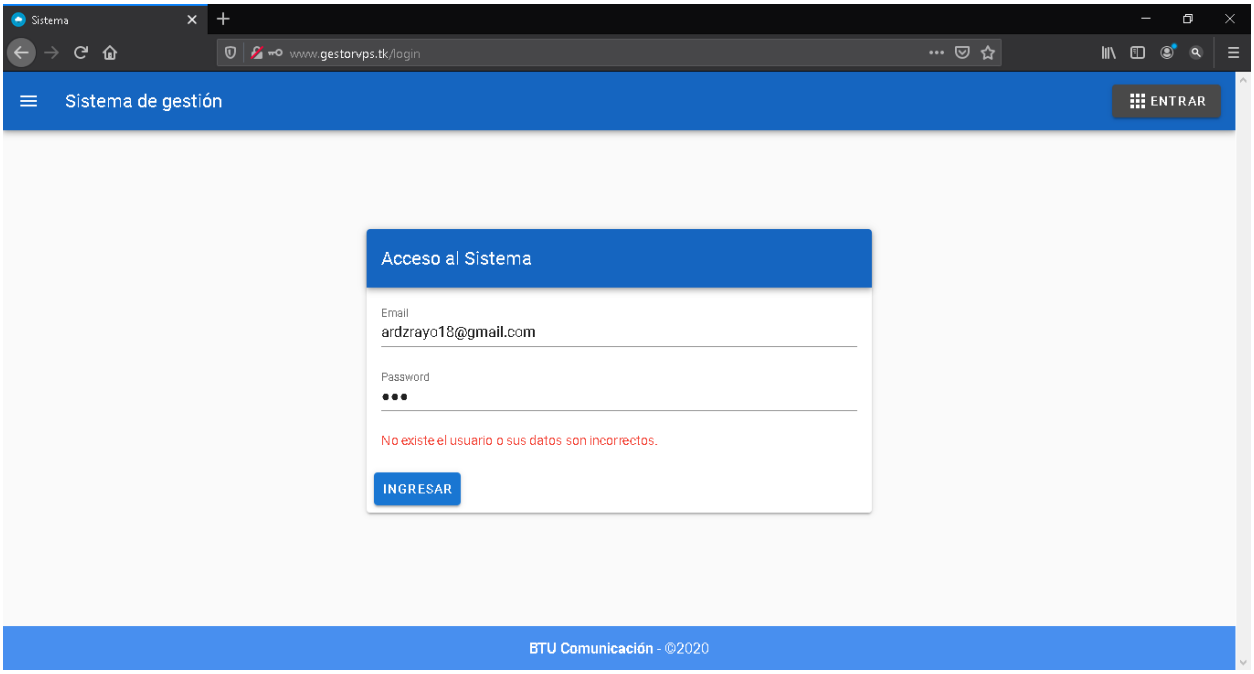

Figura 5.2 Error de acceso al sistema.

Si hemos accedido al sistema de forma correcta se nos generará un token de autentificación que nos permitirá hacer uso de la misma, este token tiene un tiempo límite de inactividad de 30 minutos, si nosotros excediéramos ese tiempo tendríamos que volver a iniciar sesión para trabajar con la página.

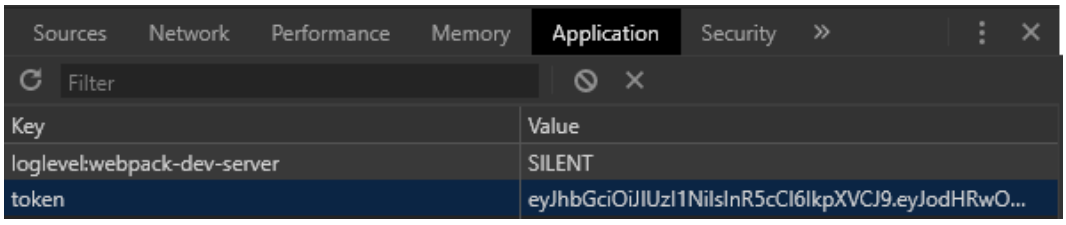

Figura 5.3 Token de acceso al sistema.

5.2 Vista principal

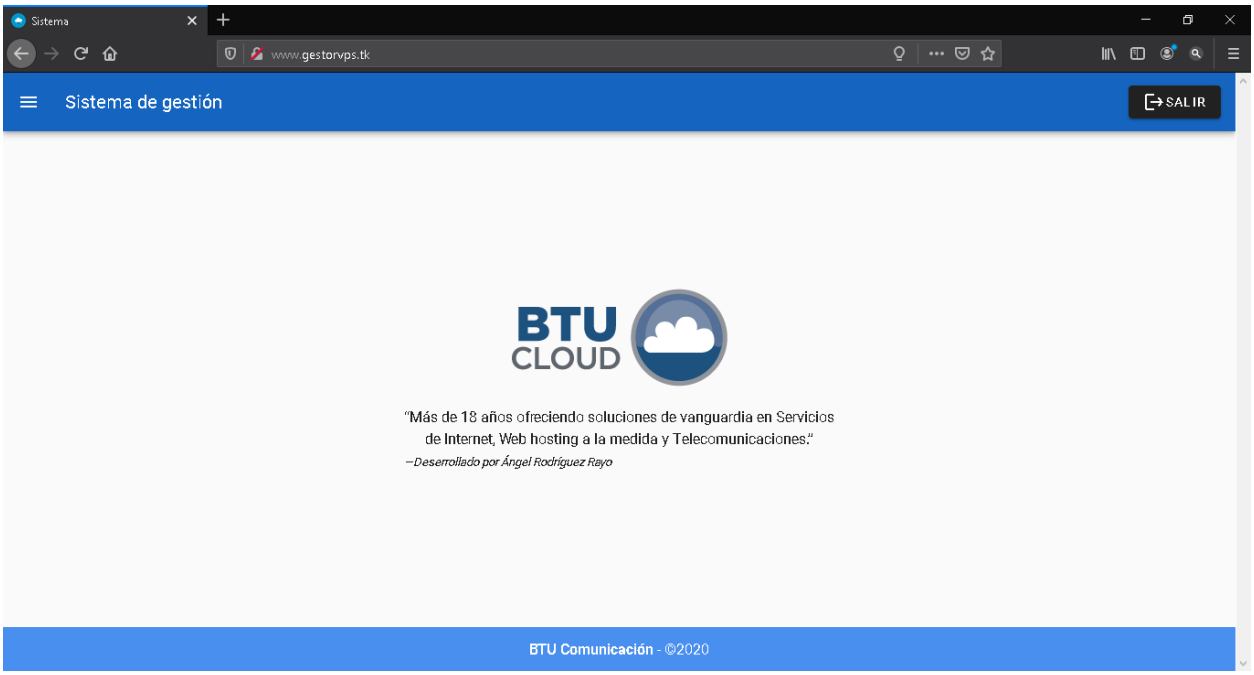

Figura 5.4 Vista principal.

La vista principal o página de inicio es aquella que se nos muestra automáticamente tras haber iniciado sesión de forma correcta, esta vista está conformada por un menú lateral izquierdo que se puede ocultar al dar clic en el botón a la izquierda del texto Sistema de gestión, del lado superior derecho podremos observar un botón que nos da la opción de cerrar sesión, botón que destruirá el token recién creado y nos regresara a la ventana de acceso al sistema, en el centro de la página podremos observar el logo de la organización acompañado de una breve descripción de los servicios que se ofrecen, en la parte inferior se cuenta con un *footer* en el cual podemos agregar información relevante de la página, como sus números de teléfono para contactarlos entre otros. El contenido del menú cambia de acuerdo al rol que tenga acceso al sistema, el menú que observamos en la figura 5.5 es el del rol de administrador.

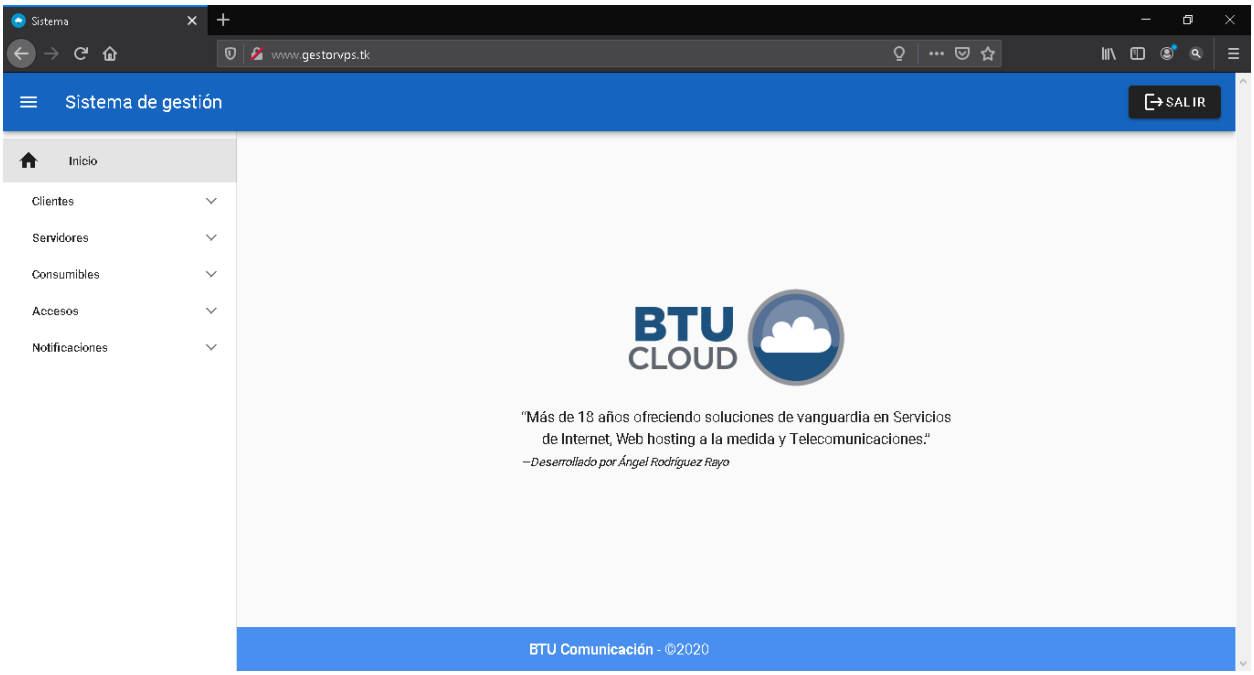

Figura 5.5 Vista principal del rol administrador.

Los roles disponibles son **Administrador**, **Soporte** y **Gerente** como se indicaron en el capítulo 4, el menú del Administrador le permite tener acceso a todo el sistema (figura 5.5), mientras que la vista de Soporte la diferencia actual con la de Administrador es el acceso al submenú Accesos, en el cual se muestran los roles y usuarios registrados en el sistema (ver figura 5.6).

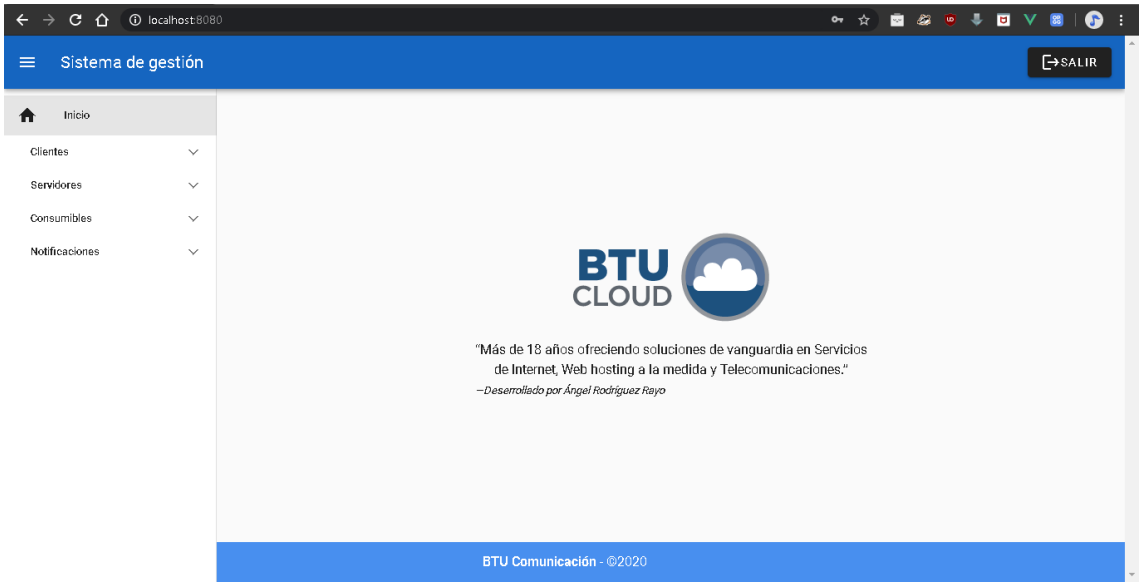

Figura 5.6 Vista principal del rol Soporte.

De igual forma existe el rol de Gerente el cual no necesita tener un acceso a los consumibles de los servidores virtuales, ya que el gerente no se encargará de realizar el registro de ellos (ver figura 5.7) y al mismo tiempo le genera una vista más limpia del sistema SYSVPS:

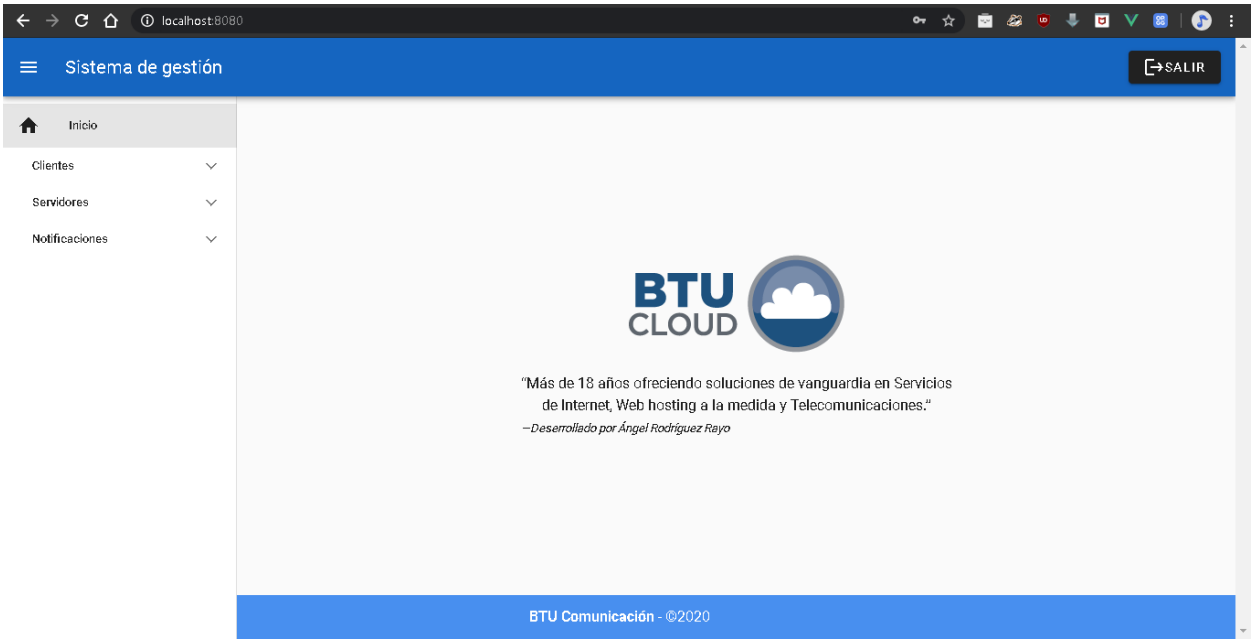

Figura 5.7 Vista principal del rol Gerente.

5.3 Sub-Menú Clientes

Dentro del sub-menú Clientes se encuentra el módulo VMClients, módulo donde se podrán visualizar los registros de los clientes de servidores virtuales privados, podremos consultarlos los registros existentes, crear nuevos registros, cambiar el estado del cliente de activo a inactivo y viceversa, este módulo está disponible para los roles administrador, soporte y gerente.

5.3.1 Módulo VMClients

Este módulo de la herramienta web corresponde a los clientes de los servidores virtuales, información necesaria para poder realizar el registro de un VPS o servidor virtual privado, ya que es el responsable de este servicio (ver figura 5.8).

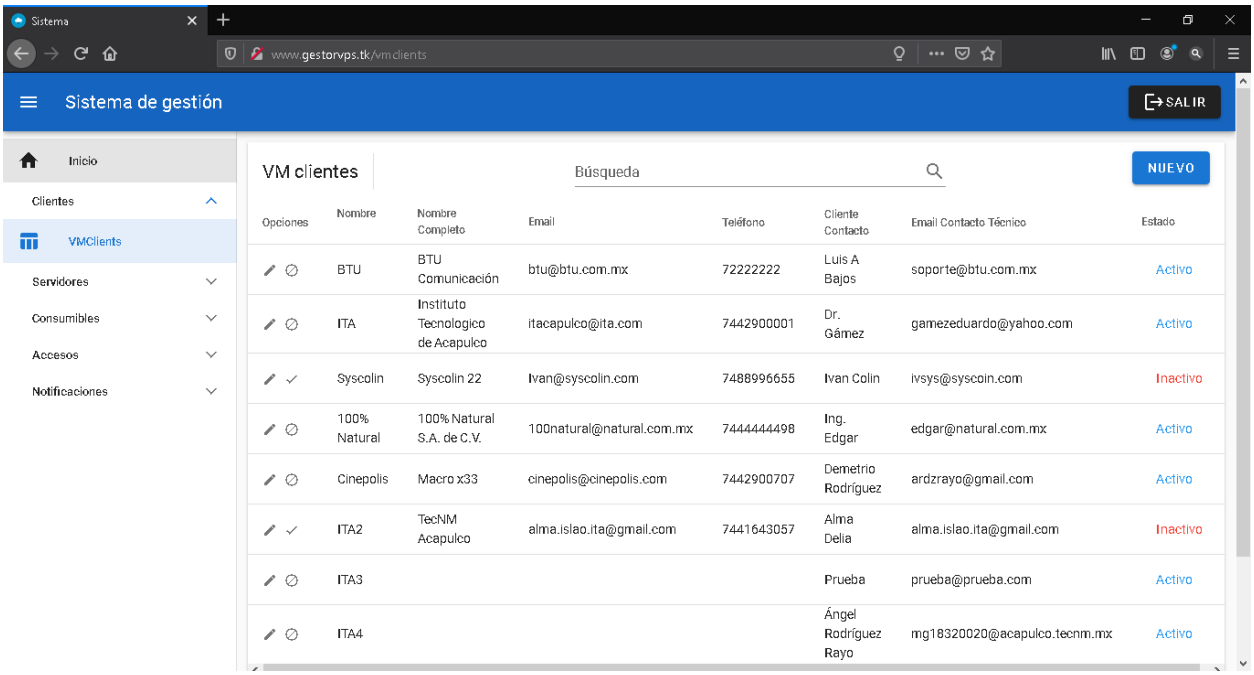

Figura 5.8 Módulo VMClients.

Como podemos observar en la figura 5.8 este módulo nos genera principalmente un listado de todos los clientes disponibles, las acciones que podemos llevar a cabo dentro de esta vista es la de ordenar de forma ascendente o descendente el nombre de la empresa o nombre completo de la

empresa cliente, generamos una barra de búsqueda en la cual podremos facilitar el encontrar un registro en específico, podemos crear un nuevo registro dando clic en el botón NUEVO, este nos abrirá un modal para realizar el llenado de datos para generar un nuevo registro (ver figura 5.9), cuenta con un campo opciones, las cuales son editar y cambiar el estado actual de un cliente (ver figura 5.10).

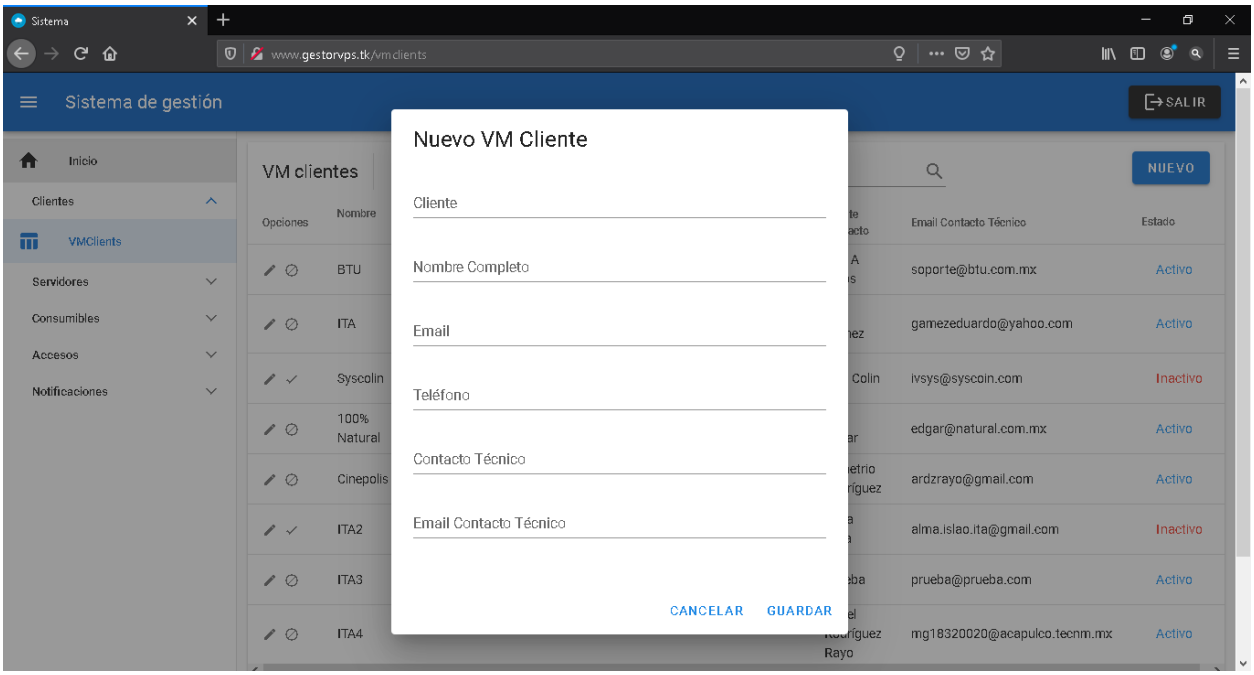

Figura 5.9 Nuevo registro de VMClient.

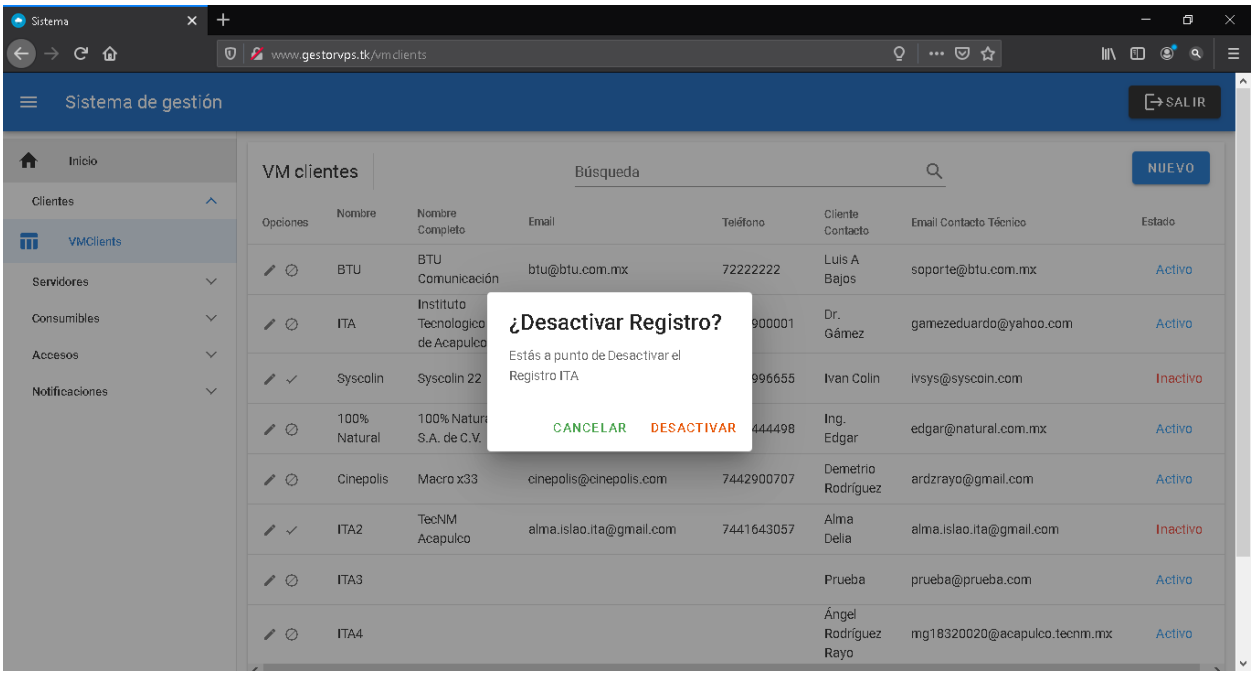

Figura 5.10 Desactivar un registro.

La opción de desactivar el registro modificará la leyenda del campo está de Activo a Inactivo, esto dentro de nuestra vista, dentro de nuestro gestor de base de datos SQL Server el nuevo valor asignado a la propiedad de estado será 0, ya que la propiedad estado es un campo de tipo **bit**.

Este módulo estará disponible para los roles: Administrador, Soporte y Gerente de la herramienta.

5.4 Sub-Menú Servidores

Dentro del sub-menú Servidores se encuentran los módulos VM List y el módulo Pools, VM List contiene los registros de los servidores virtuales públicos y el módulo Pools hace referencia a los servidores físicos donde son almacenados un grupo de servidores virtuales, servidores de alta capacidad de procesamiento que se encarga de administrar el rendimiento de los VPS.

5.4.1 Módulo VM List

Este módulo de la herramienta web corresponde a los registros de servidores virtuales privados, VM haciendo referencia a Virtual Machine, este módulo está disponible para los roles administrador, soporte y gerente, a pesar de que los 3 roles tienen acceso al módulo para la lectura de datos no todos tienen acceso a realizar modificaciones dentro del mismo, los únicos roles que pueden realizar todos los procedimientos son el rol de administrador y el rol de soporte.

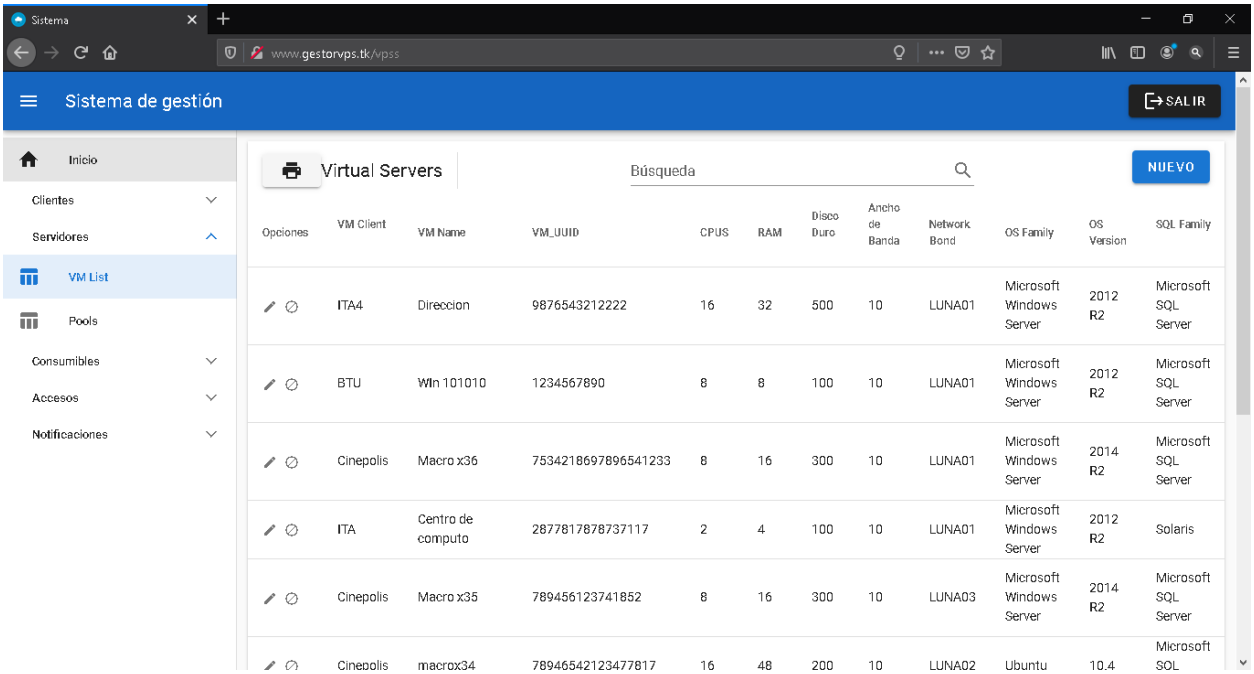

Figura 5.11 Módulo VM List.

5.4.2 Módulo Pools

Este módulo del sistema web corresponde a los registros de los pools que posee la compañía, cuya función es la de almacenar y virtualizar los servidores virtuales privados, los roles que tienen acceso a este módulo son administrador, soporte y gerente, donde el gerente solo tiene acceso a la lectura del método, los roles restantes tienen acceso a los procedimientos de actualizar, crear y desactivar o activar un registro.

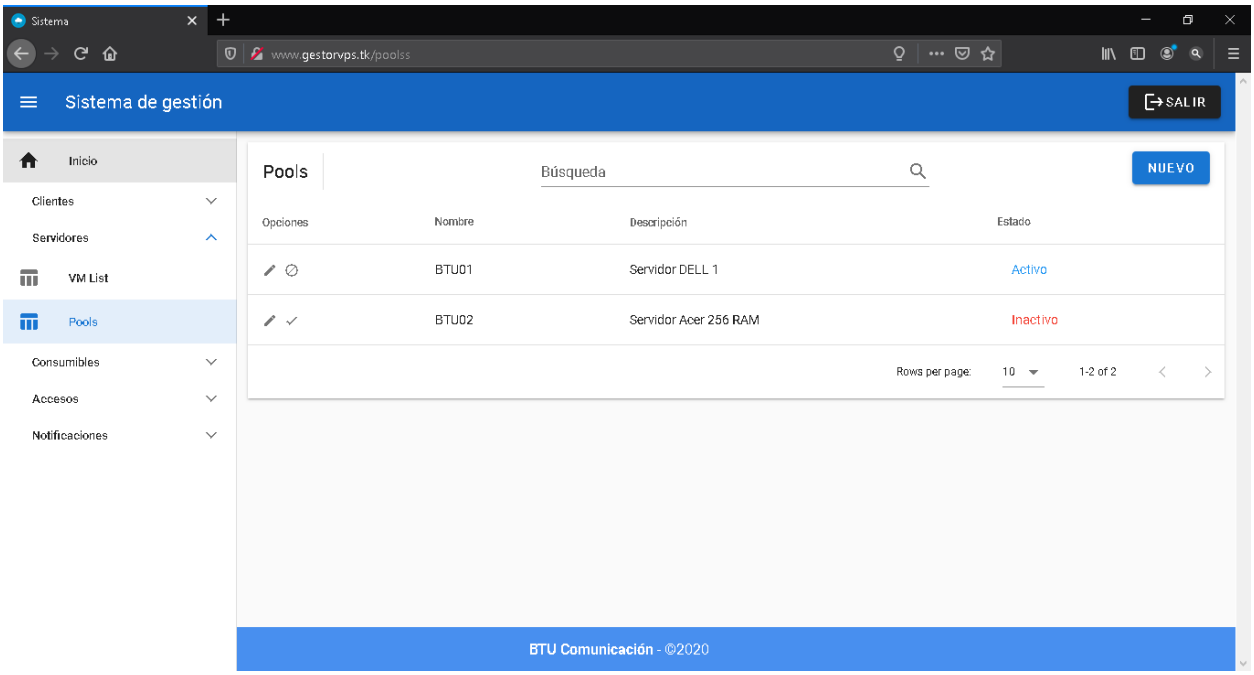

Figura 5.12 Módulo Pools.

5.5 Sub-Menú Consumibles

Los consumibles son un conjunto de características que intervienen al momento de realizar un registro de servidor virtual privado, actualmente solo se contienen consumibles para servidores virtuales, en trabajos a futuro dentro de este sub-menú se podrían almacenar otro tipo de consumibles para otros servicios. Los roles que tienen acceso a este sub-menú son los de administradores y soporte.

5.5.1 Módulo Networtk Bond

Este módulo de la aplicación web almacena los Network bonds disponibles para los servidores virtuales, un network bond es una interfaz de red virtual que se utiliza para distribuir de forma óptima la carga de datos que tiene que recibir el router de la compañía. Los roles que tienen acceso a los procedimientos de lectura y escritura son los de administrador y soporte.

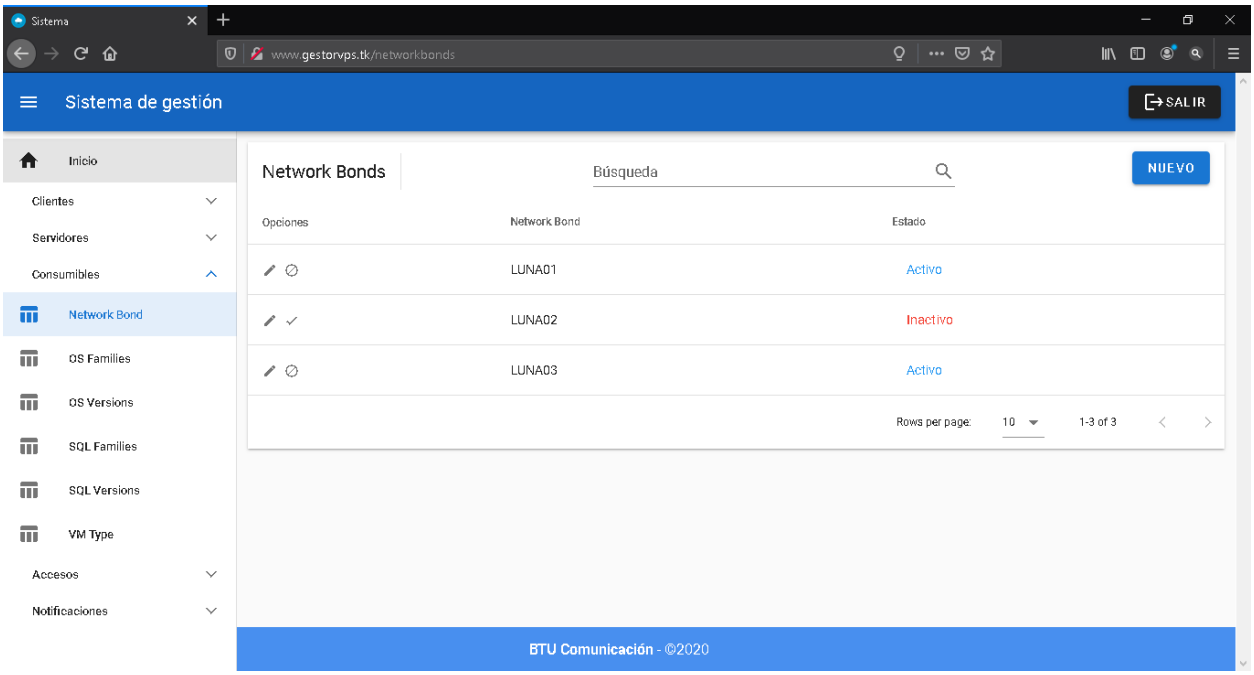

Figura 5.13 Módulo Network Bond.

5.5.2 Módulo OS Families

Este módulo de la herramienta web nos muestra los registros de sistemas operativos con los que cuenta la empresa para ser instalados en los servidores virtuales, en este módulo se pueden agregar nuevos registros y desactivar los sistemas que se consideren obsoletos, esta información va relacionada a la tabla VPS de la base de datos y es un requerimiento importante al momento de realizar soporte técnico. Los roles que tienen acceso a este módulo son los de administrador y soporte.

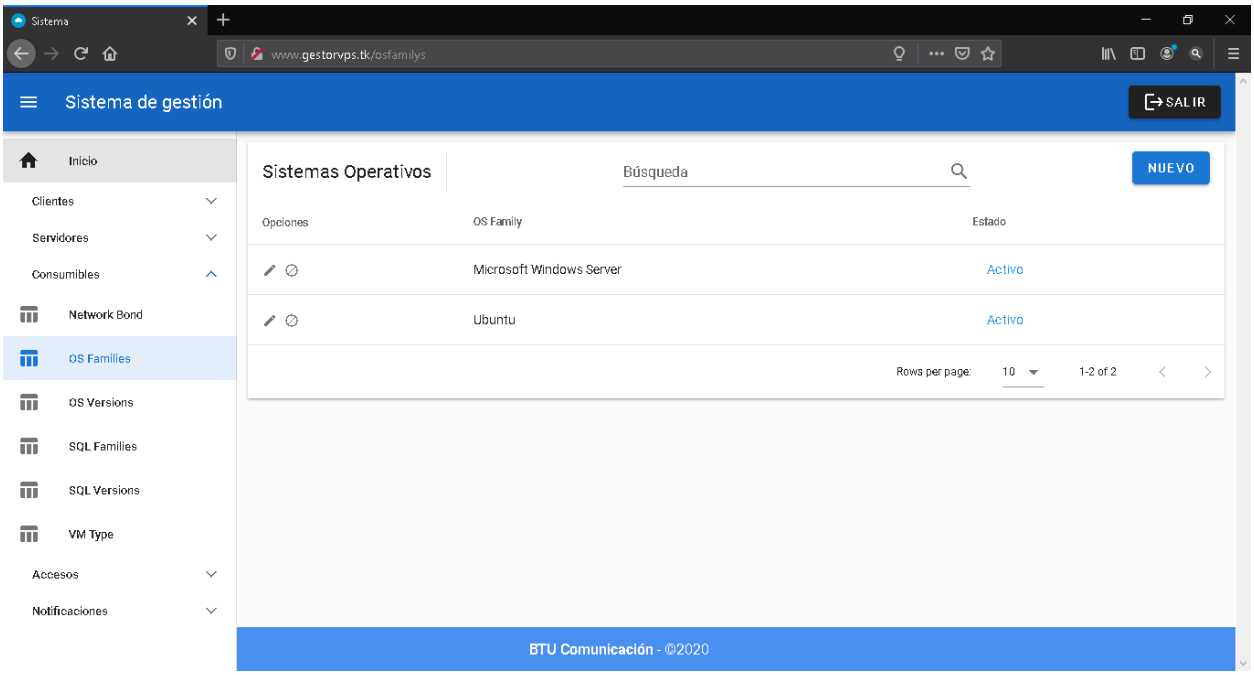

Figura 5.14 Modulo OS Families.

5.5.3 Módulo OS Versions

El módulo OS Versions está relacionado con el módulo OS Families ya que se refiere a la versión de los sistemas operativos existentes, por ejemplo dentro de los sistemas operativos que ofrece la compañía esta Microsoft Windows Server, de este sistema operativo existe la versión 2012 y la versión 2012 R2 o 2014 y la versión 2014 R2 se maneja una gran variedad debido a que las organizaciones que requieren de un servidor virtual ya tenían instaladas sus aplicaciones en determinada versión, para brindar de una manera más ágil atención al cliente se detalla a mayor manera el registro de un VPS. Los roles que tienen acceso a este módulo son los de administrador y soporte.

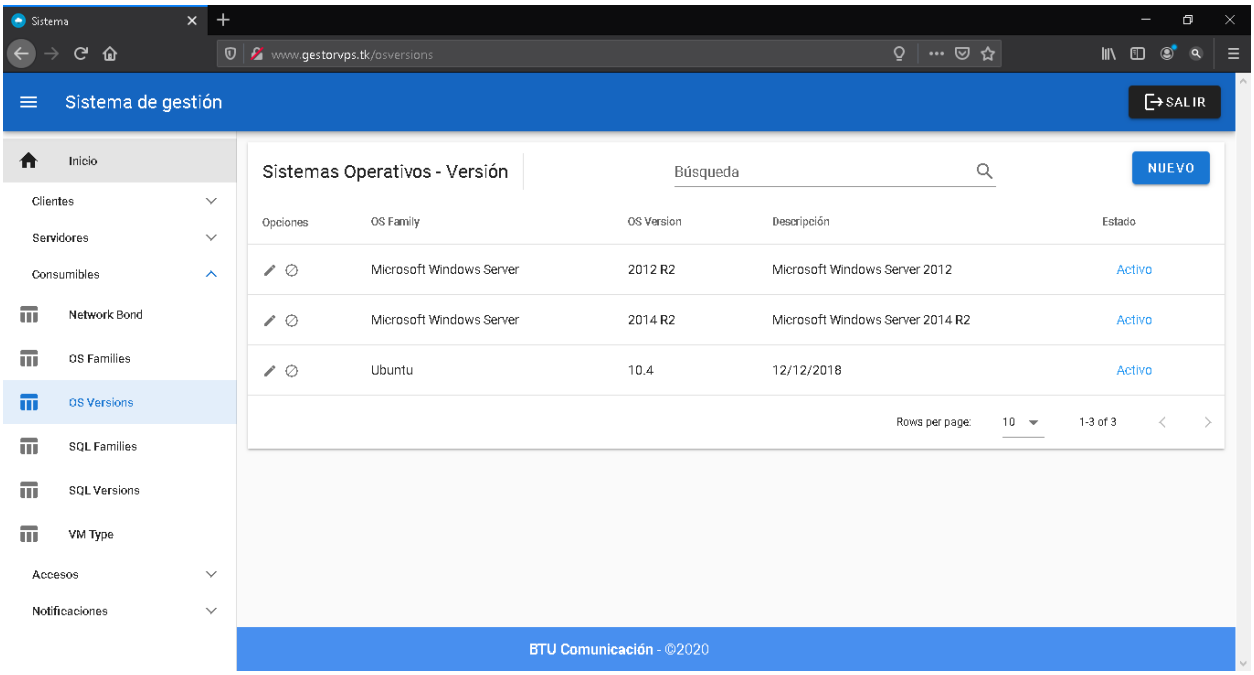

Figura 5.15 Módulo OS Versions.

5.5.4 Módulo SQL Families

Este módulo de la herramienta nos da acceso a los diferentes gestores que maneja la compañía como complemento a los servidores virtuales privados, la instalación de un gestor de base de datos es un complemento que puede solicitar un cliente si así lo desea, debido a que la organización maneja un control de cuantas licencias mantiene en uso mensualmente se lleva un registro dentro del sistema para facilitar su administración. Los roles que tienen acceso a este módulo son los de administrador y soporte.

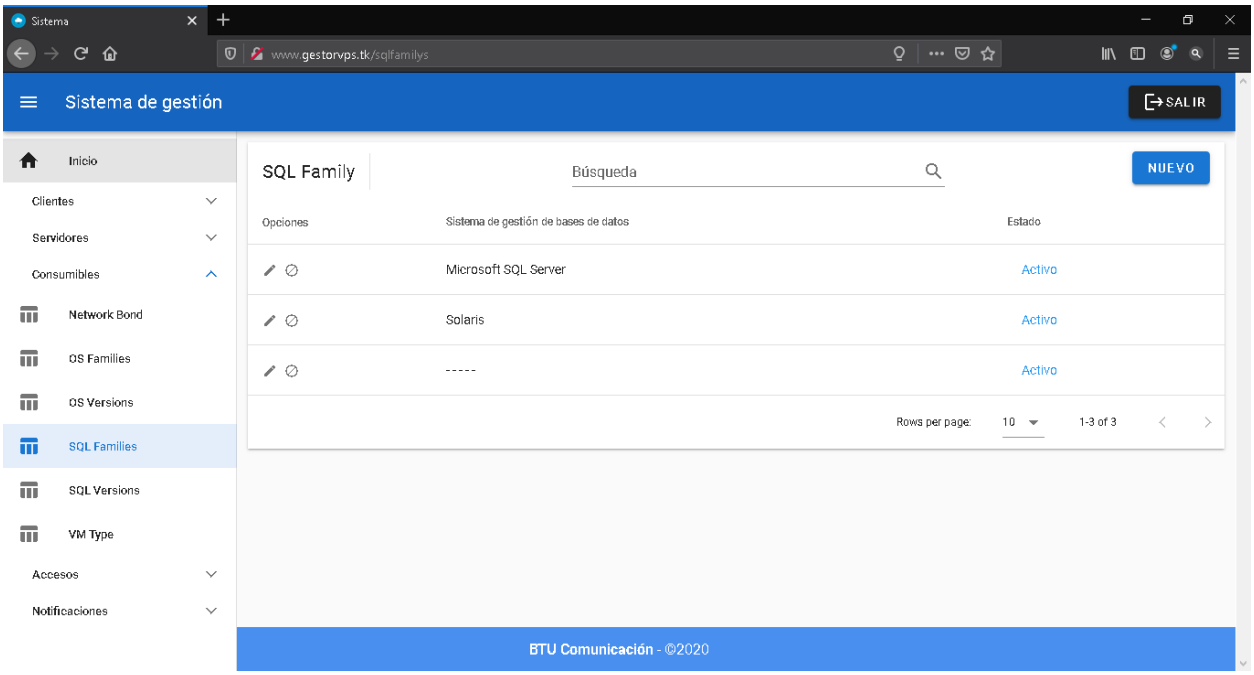

Figura 5.16 Modulo SQL Families.

5.5.5 Módulo SQL Versions

Este módulo tiene relación con el módulo SQL Families, su relación es que en este se almacenan las versiones de los gestores de base de datos, como por ejemplo dentro del gestor SQL Server existen muchas versiones, como la versión Enterprise y la versión Web, entre otras. Los roles que tienen acceso a todos los procedimientos dentro del módulo son los de administrador y soporte.

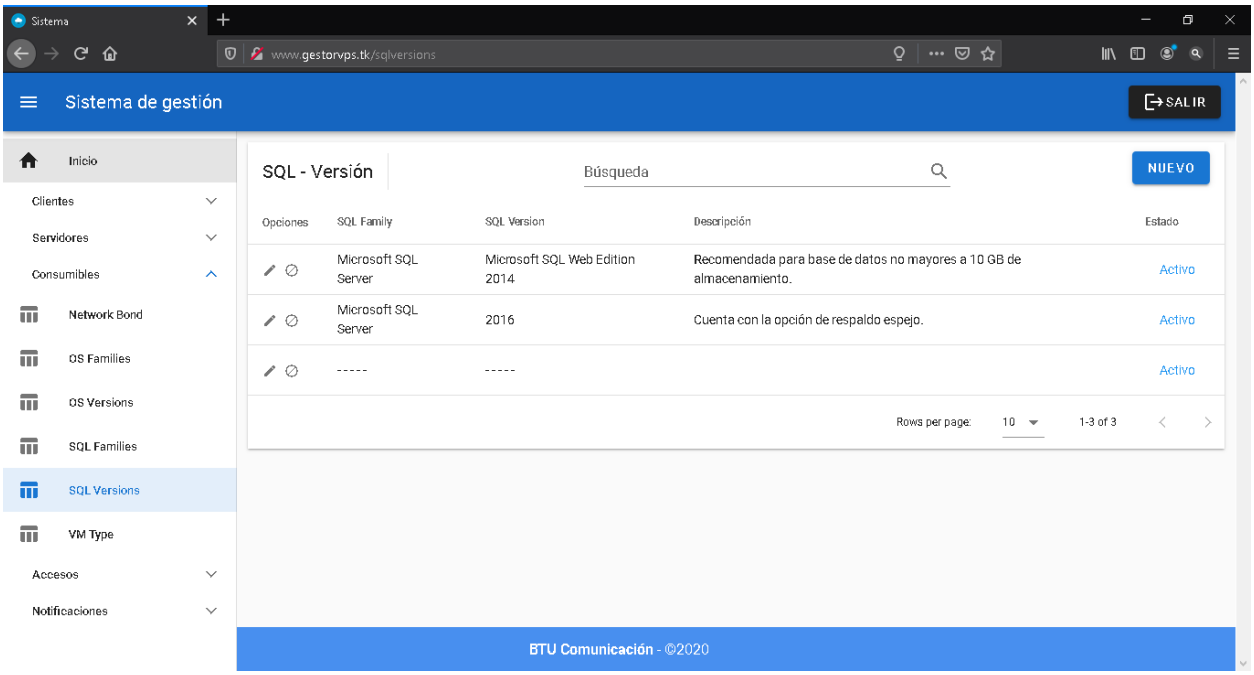

Figura 5.17 Modulo SQL Version.

5.5.6 VM Type

Este módulo de la herramienta web almacena los diferentes tipos de máquinas virtuales que puede proporcionar la organización a sus clientes, como por ejemplo un tipo de ellos sería renta y otro demo, como estos existen otros tipos de máquinas virtuales que puede proporcionar a la organización a sus clientes.

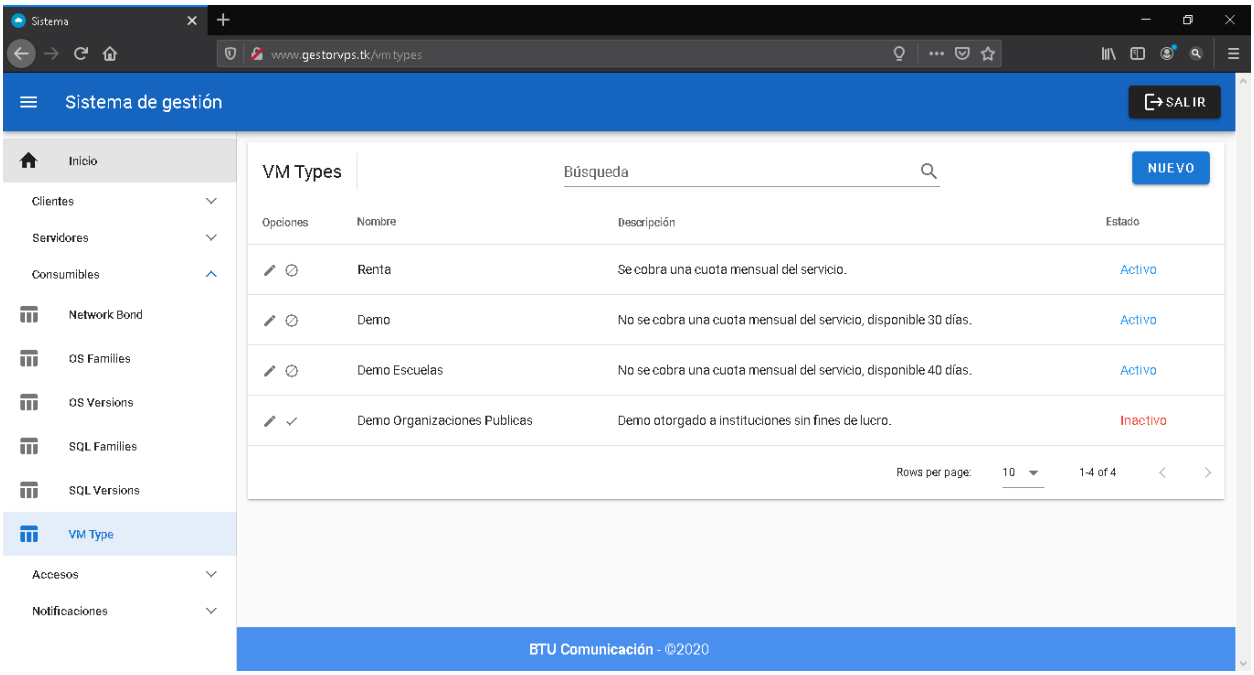

Figura 5.18 Módulo VM Type.

5.6 Sub-Menú Accesos

Dentro de este sub-menú se encuentra el módulo Roles y el módulo Usuarios. A este módulo solo tiene acceso el rol de administrador, debido a la importancia que conlleva gestionar las cuentas de usuario que pueden entrar al sistema.

5.6.1 Módulo roles

Este módulo de la herramienta web nos muestra los roles existentes dentro del sistema, dentro de este módulo solo podremos leer los datos de los registros de rol existentes y visualizar su estado. El rol que tiene acceso a este módulo es el de administrador.

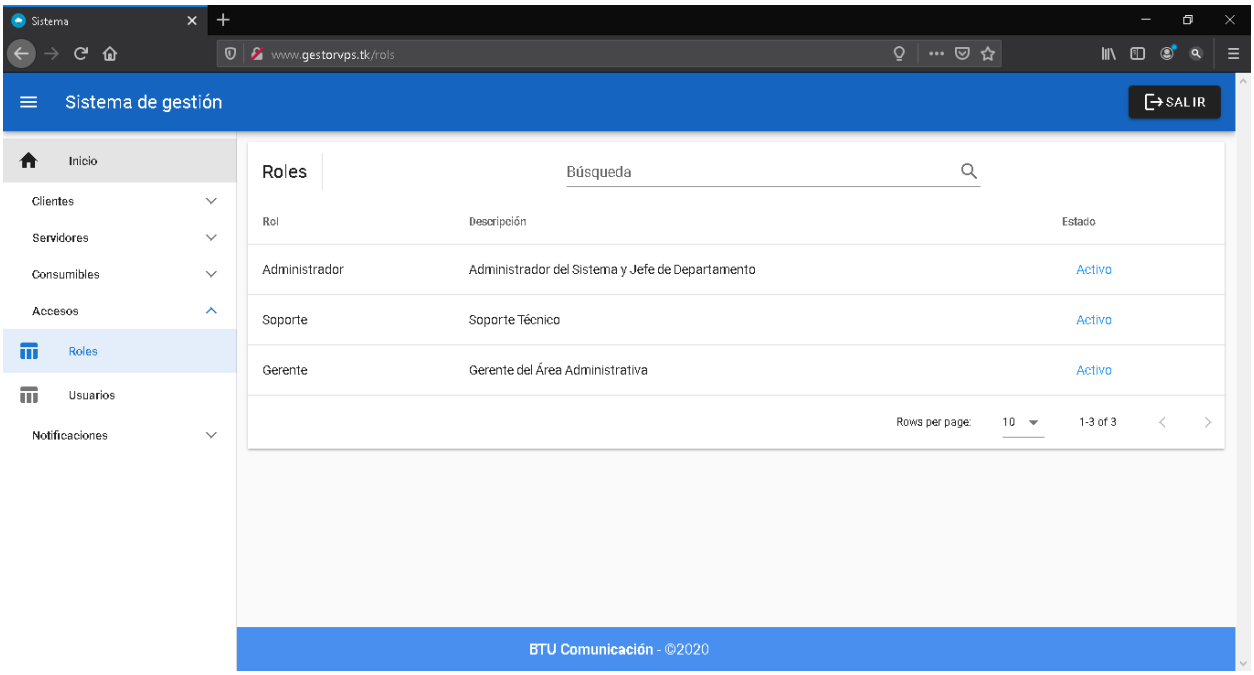

Figura 5.19 Módulo Roles.

5.6.2 Módulo Usuarios

Este módulo de la herramienta web corresponde a los usuarios que se encargarán de manejar la aplicación, agregando, editando y activando o desactivando registros. El rol que tiene acceso a este módulo es el de administrador debido a la importancia que tiene el poder visualizar a todos los usuarios existentes y modificar su contraseña actual en caso de que un usuario pierda el acceso.

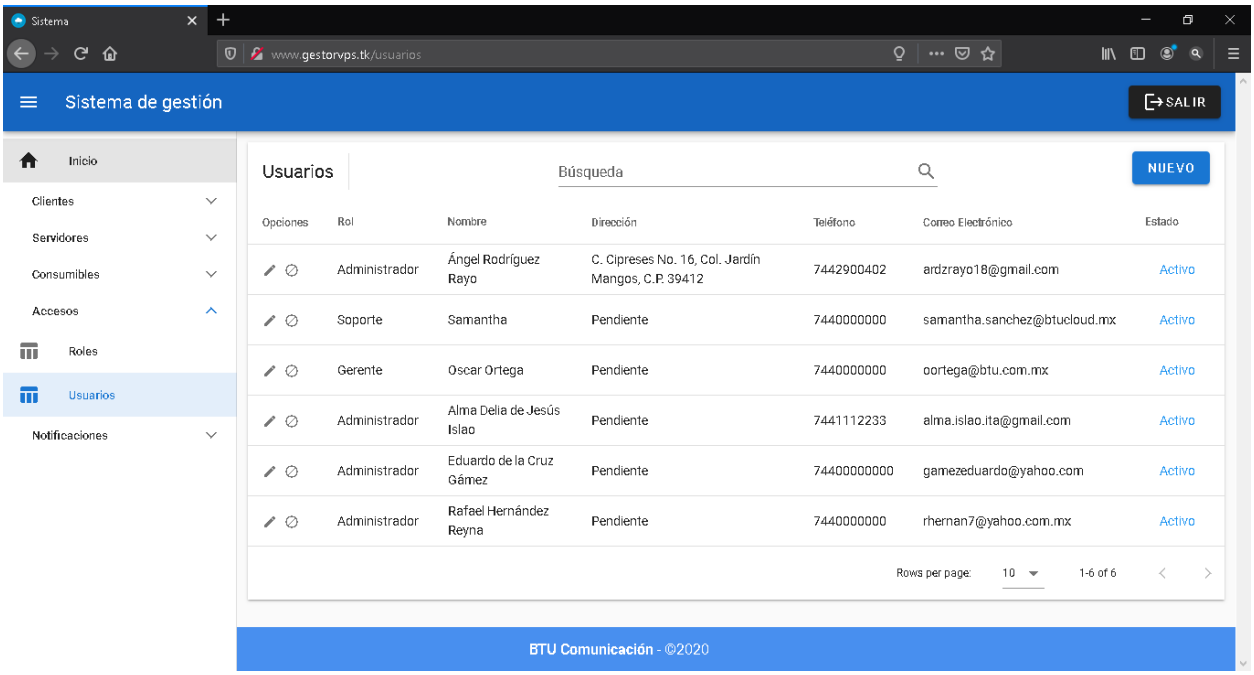

Figura 5.20 Módulo Usuarios.

5.7 Sub-Menú Notificaciones

Este módulo de la aplicación web es referente a las notificaciones vía correo electrónico que son enviadas a los clientes que se encuentren pendientes de realizar el pago de sus servicios. Los roles que tienen acceso a esta vista son los de administrador, soporte y gerente.

5.7.1 Configuración de notificaciones

Este módulo llamado notificar de la aplicación web nos permitirá realizar una modificación a la configuración del envió de notificaciones a los clientes, actualmente BTU Comunicación cuenta con un periodo de notificación ya definido que se respetará en esta aplicación Web, la modificación que podrá realizarse es si a ese registro en esta vista será notificado, en esta vista se creó una tabla a partir de la tabla clientes y servidores virtuales. Los roles que tienen acceso a este módulo son los de administrador, soporte y gerente.

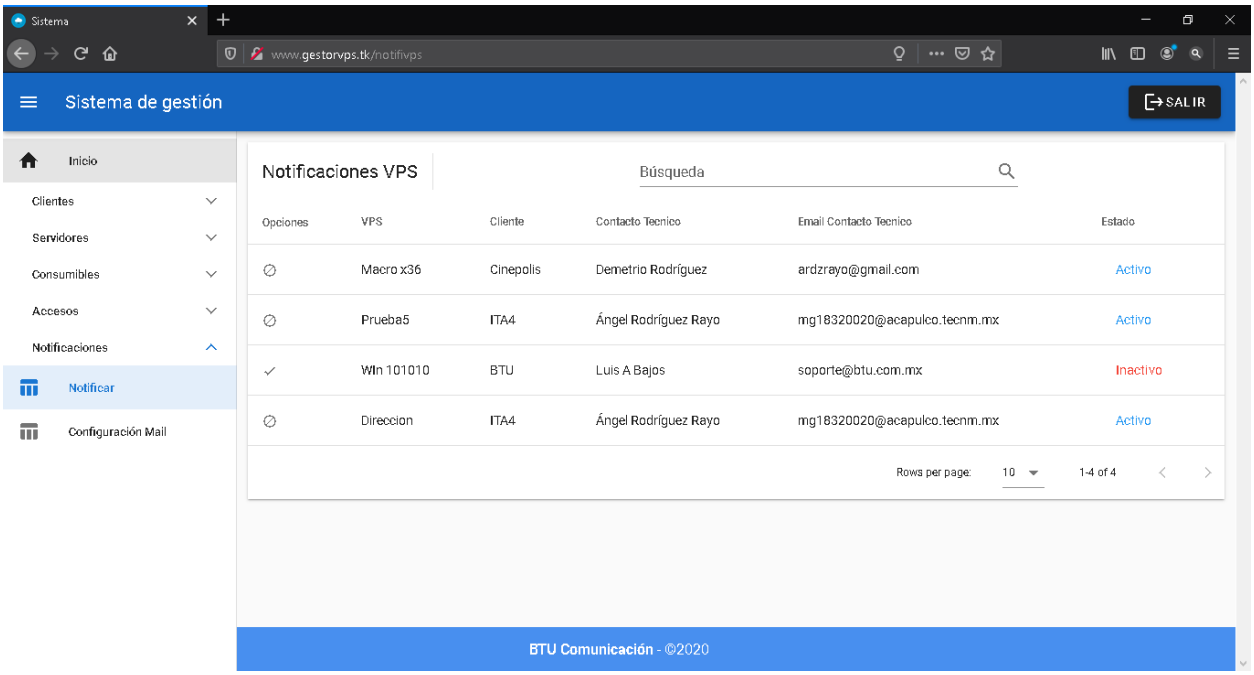

Figura 5.21 Modulo Notificar.

Para logar el funcionamiento de esta vista se implementaron seis **triggers** o **disparadores** en la base de datos, estos disparadores se ejecutan cuando se realizan diversas acciones, el principal, el cual consiste en la inserción de los datos a esta vista, esté se dispara cuando se realiza un registro dentro de la tabla **VPS**, tomando los datos necesarios de esta tabla y los datos del cliente que están relacionados a ese servidor, todo relacionado por su llave principal. Los **trigger**s restantes se ejecutan al realizar una modificación en alguno de los campos de esta vista, el **trigger** a destacar es el que se ejecuta al cambiar el estado de un servidor virtual, ya que, si un servidor virtual se suspende, de forma automática se deshabilitaran sus notificaciones vía correo electrónico.

Mientras el registro de notificación este Activo se realiza una notificación vía correo electrónico en las fechas establecidas, las fechas establecidas en esta aplicación web son una el día 10 de cada mes y el día 12. El correo que podrá visualizar el cliente será similar al que se muestra en la figura 5.22.

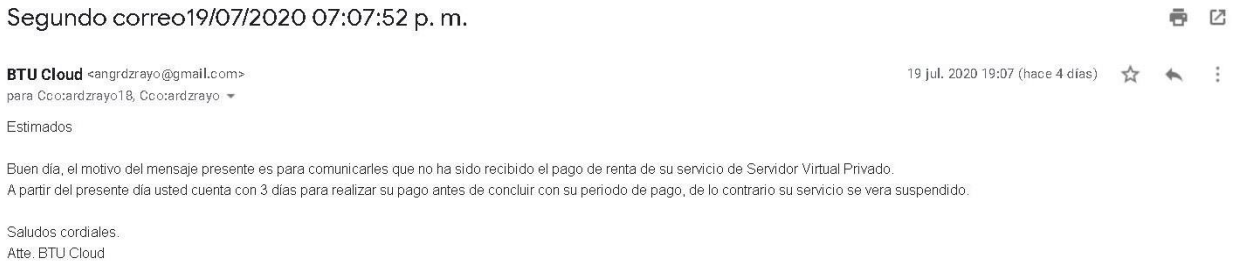

Figura 5.22 Correo Electrónico proporcionado por la aplicación web.

5.8 Conclusiones

A manera de conclusión del trabajo desarrollado en la presente tesis, se cumplió con los objetivos específicos planteados en la introducción del trabajo. Se analizó el marco de trabajo para la gestión de proyectos de desarrollo de software basado en la metodología RUP y Scrum debido a que la herramienta propuesta fue presentada para su análisis en múltiples entregas del producto. Esto permitió un mejor entendimiento de los diferentes procesos de la misma e identificar de mejor forma todos los aspectos que podrían optimizarse. Posteriormente, haciendo uso de las herramientas de desarrollo descritas en el marco teórico y soportado por el diseño presentado en el capítulo 3 se codificaron funciones de la herramienta propuesta, relacionando cada uno de los componentes (módulos, vistas, viewmodels y controladores) de la herramienta para un correcto intercambio de información. Por último, se probaron cada una de las funciones desarrolladas de la herramienta de gestión administrativa, comprobando que las transacciones en la base de datos se realizaban correctamente.

Por otra parte, en la presentación de la funcionabilidad y operación del sistema de gestión administrativa para la notificación de clientes con servidores virtuales activos al responsable del área de sistemas computacionales de la empresa BTU del municipio de Acapulco se demostró que utilizando herramientas de nueva generación para desarrollar software y utilizando patrones de diseño de software, también como repositorios repletos de librerías de desarrollo de software que simplifican el tiempo de codificación y la abundante documentación de estas, se pueden crear sistemas de gestión administrativa a medida de la organización de tal manera que puedan agilizar el periodo de tiempo para realizar actividades que son importantes para la organización y brindar un mejor servicio a sus clientes.

5.9 Trabajos a futuro

Una de las aportaciones que se le pueden agregar al presente trabajo en la parte del desarrollo de software es agregar la funcionalidad de suspender de forma automática el servidor que mantenga activa sus notificaciones al terminar el periodo de pago permitido por la compañía BTU Comunicación, esta acción se puede manejar utilizando diferentes alternativas, una es usando la propiedad UUID que genera el servidor de virtualización Xen Server, donde podremos apagar el equipo, otra de las opciones es la de cortar la conexión del servidor virtual con exterior y solo mantenerlo trabajando de forma local, cada servidor cuanta con una IP privada que se administra de forma local en la empresa y un IP publica, la cual hace un enlace con la IP privada y le da acceso al usuario exterior de tener acceso al servidor virtual, la operación a realizar seria desactivar este vínculo de conexión, de esta forma el cliente no podría acceder al servicio.

Aditivo a esto es posible agregar otros servicios que proporciona la empresa a sus clientes, permitiendo así una mejor gestión de ellos. Otro de las posibles opciones que se podrían emplear sería la de crear una aplicación móvil para teléfonos inteligentes, esto debido a que nuestra herramienta web se configuro como una herramienta API, debido a esto los programadores móviles podrían utilizar el back-end existente sin tener que volver a crear la lógica de negocios en un nuevo entorno.

Referencias

Axios, P. p. (16 de 03 de 2017). *https://laesporadelhongo.com/.* Obtenido de https://laesporadelhongo.com/wp-content/cache/page_enhanced/laesporadelhongo.com/primeros-pasos-con-axios/_index.html_gzip

Blankenship, J., Bussa, M., & Millett, S. (s.f.). *Pro Agile .Net Development with Scrum.* Apress.

Bosch, J. (2000). *Design & Use of Software Architectures.* Addison-Wesley.

Chiaretta, S. (2018). *Front-end Development with ASP.NET Core, Angular, and Bootstrap.* Indiana: Wrox A Wiley Brand.

Cohen, M. (2004). *User Stories Applied for Agile Software Develoment.* The Addison-Wesley Signature Series.

Enterprise Architect. (22 de Noviembre de 2019). Obtenido de wikipedia: https://en.wikipedia.org/wiki/Enterprise_Architect_(software)

Eslava, V. (2011). *El Nuevo PHP. Conceptos Avanzados, Editorial.* España: Bubok Publishing.

Ferguson, J., Patterson, B., Beres, J., Boutquin, P., & Gupta, M. (2003). *La biblia de C#.* Madrid, España: ANAYA Multimedia.

Galeano Arenas, J. G. (09 de Enero de 2016). "Te atiendo" Plataforma integral de atención a la población del municipio de Rionegro. Catalunya, España.

González, P. R. (Septiembre de 2008). Estudio de la aplicación de metodologías ágiles para la evolución de productos software. Madrid, España.

Lock, A. (2018). *ASP.NET Core in Action.* United States of America: Manning Publications Co.

McFedries, P. (2017). Agile Development Spawns. *IEE SPECTRUM*.

Microsoft SQL Server. (1 de Mayo de 2018). Obtenido de wikipedia: https://es.wikipedia.org/wiki/Microsoft_SQL_Server

Microsoft Visual Studio. (14 de Octubre de 2019). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Microsoft_Visual_Studio

Node.js. (1 de 03 de 2020). *Wikipedia.* Obtenido de https://es.wikipedia.org/wiki/Node.js

Pressman, R. (2010). Diseño basado en patrones. En R. Pressman, *Ingeniería del Software: Un enfoque práctico* (págs. 295-316).

Refsnes Data. (2020). Obtenido de w3school.com.

Resig, J. (26 de Agosto de 2006). *Jquery*. Obtenido de Jquery: https://jquery.com

Rumbaugh, J., Jacobson, I., & Booch, G. (2000). *El lenguaje unificado de modelado. Manual de referencia.* Madrid: Addison Wesley.

Schwaber, K. B. (2006). *Agile Software Develoment with SCRUM.* Conchango.

Schwaber, K., & Sutherland, J. (2013). *La Guía de SCRUM.*

Sommerville, I. (2011). Administración de un proyecto ágil. En I. Sommerville, *Ingeniería de Software* (págs. 72-74). Pearson.

Sommerville, I. (2011). *Ingenieria de Software.* México: Pearson educacion.

Sommerville, I. (2011). *Ingeniería de Software.* México : Pearsón Educación .

Sommerville, I. (2016). *Ingenieria de Software.* México.

Tahuiton Mora, J. (2011). *Arquitectura de software para aplicaciones Web.*

Vue.js. (2020). *Vue.js.* Obtenido de Vuejs.org: https://es.vuejs.org/v2/guide/