ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO WEB PARA COMUNICACIÓN EN TIEMPO REAL, QUE PERMITA REALIZAR VIDEO LLAMADAS, TRANSFERENCIA DE ARCHIVOS Y CHAT, MEDIANTE EL USO DE WEBRTC

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE LA INFORMACIÓN

CARLOS ALFONSO MENDOZA QUICHIMBO

DIRECTOR: Ing. Andrés Reyes MSc.

CODIRECTOR: Ing. Julio Caiza MSc.

Quito, Agosto 2016

DECLARACIÓN

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

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____

Carlos Alfonso Mendoza Quichimbo

CERTIFICACIÓN

Certificamos que el presente trabajo fue desarrollado por Carlos Alfonso Mendoza Quichimbo, bajo nuestra supervisión.

Ing. Andrés Reyes, MSc.
DIRECTOR DEL PROYECTO

Ing. Julio Caiza, MSc.
CODIRECTOR DEL PROYECTO

AGRADECIMIENTOS

Primeramente quiero agradecer a Dios, por permitirme vivir esta grandiosa etapa de vida y culminar exitosamente mis estudios universitarios.

Agradezco a toda mi familia en especial a mis tías Marlene y Marisol, quienes con sus palabras han transmitido en mí, esos deseos de superación y me han fortalecido en momentos de flaqueza, gracias de todo corazón.

Agradezco de manera especial a mis hermanas Cristina, Gabriela y Belén por ser más que hermanas, amigas que Dios ha puesto en mi camino y que sepan que pueden contar conmigo en las buenas y en las malas, agradezco mucho su apoyo. A Cristina que ha sido un cimiento para la construcción de mi vida profesional, gracias a su gran ejemplo, dedicación y responsabilidad que me han llevado cada día a admirarla más.

Muchas gracias a mis amigos, amigas, compañeros y compañeras, con aquellos que tuve la gran suerte de compartir las aulas y con los que no de igual manera; que me brindaron la mano cuando más lo necesitaba y compartimos muchas experiencias de vida; momentos gratificantes los cuales siempre van a estar en mí como un hermoso recuerdo.

Agradezco de manera especial a la Escuela Politécnica Nacional, y por medio de ella, a mis maestros y maestras, quienes fueron los precursores para formar en mí una persona útil para la sociedad.

De igual manera a la MSc Jenny Tubón, trabajadora social de la EPN, por todos sus consejos y por su valiosa amistad, que me han servido de mucha ayuda y superación en momentos difíciles de mi vida.

A mis tutores Ing. Andrés Reyes e Ing. Julio Caiza los cuales me apoyaron con su tiempo y paciencia para la realización de este trabajo.

"El agradecimiento es la memoria del corazón"

Lao-Tse

DEDICATORIA

El esfuerzo, lucha, tiempo y dedicación entregados a lo largo de mi vida estudiantil, que han permitido que el día de hoy pueda redactar estas palabras no serían posibles si no hubiera estado conmigo la más bondadosa persona que Dios me ha dado, aquella persona que me ha brindado su amor incondicional, su paciencia y cariño; persona que con su infinita dedicación, perseverancia y enseñanzas han hecho de mi la persona que ahora soy.

Mujer que me llena de mucho orgullo y sé que todo el tiempo del mundo no bastará para agradecerte por todo lo que me has dado, ya que este logro que llevo a cabo es gracias a ti.

Gracias madre por ser la principal promotora de mis sueños, gracias por confiar, creer en mí y en mis expectativas, gracias por haber sido mi primera maestra y haber estado dispuesta a acompañarme en cada larga y agotadora noche de estudio. Por siempre desear y anhelar lo mejor para mi vida; por cada consejo y cada palabra que me supieron guiar en la escuela de la vida.

Con mucho cariño y amor esta obra va dedicada a ti, gracias mamá...!!

"Es increíble todas las virtudes que Dios ha puesto en ti Mamá,

Gracias por todo lo que me has enseñado"

Carlos Mendoza

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En el primer capítulo, se presenta una revisión sobre los principales conceptos de WebRTC, se describe su arquitectura, el conjunto de las interfaces de aplicación que la conforma y una breve reseña histórica.

También se realiza una introducción de JavaScript y de la quinta versión de HTML, se describe como se organizan las etiquetas en la página web y se realiza mención de sus librerías externas. Finalmente se realiza un estudio del *framework* de XSockets y del patrón MVC.

En el segundo capítulo, básicamente se divide en dos secciones, en la primera se realiza el análisis de requerimientos en una empresa que brinda servicios de telecomunicaciones.

En la segunda sección se realiza el diseño de la aplicación, se presentan el diagrama de la base de datos, diagramas de actividades y el diagrama de clases. Finalmente se presentan las vistas del sistema.

El tercer capítulo también ha sido dividido en dos secciones, en la primera sección se ha enfocado en realizar la implementación de cada uno de los módulos de la aplicación web y se presentan las vistas obtenidas y porciones de código relevantes del sistema. En la segunda sección se realiza una evaluación de los módulos para verificar el correcto funcionamiento de la aplicación web.

El cuarto capítulo presenta las conclusiones que se obtuvieron durante el desarrollo del presente proyecto y las recomendaciones para la realización de trabajos futuros.

PRESENTACIÓN

WebRTC se encuentra siendo impulsado principalmente por Google, debido a los altos beneficios que esta tecnología de comunicación representa. Entre las mayores ventajas está la de realizar una comunicación en tiempo real basado en un navegador Web, además un incremento en la productividad, debido a una reducción en desplazamientos innecesarios que realizan los usuarios para organizar una reunión de trabajo.

De igual manera, al ser una tecnología de código abierto, permite a los desarrolladores web, repotenciar y fortalecer sus características, ya que WebRTC utiliza interfaces de aplicación desarrolladas a partir de código JavaScript.

El propósito del presente proyecto es el de ofrecer una aplicación web que permita realizar video-llamadas, chat y transferencia de archivos sin la necesidad de depender de *plugins* externos para su funcionamiento, característica soportadas por WebRTC y además ofrecer una aplicación responsiva, que se ajusta automáticamente al tamaño de pantalla en la que se visualiza.

El sistema está compuesto por el módulo de autenticación que permite ingresar al sistema, el módulo de administración de usuarios y módulo de administración de grupos; que habilitan las opciones para crear, modificar y eliminar usuarios; finalmente el módulo de comunicación permite interactuar a un usuario con otro.

1 CAPÍTULO 1: FUNDAMENTOS TEÓRICOS

1.1 INTRODUCCIÓN

La Comunicación en Tiempo Real (RTC) a través de la Web, también denominada WebRTC, añade nuevas funcionalidades a los navegadores web. WebRTC permite que los navegadores interactúen directamente con otros navegadores, dando paso a un número amplio de arquitecturas, las más comunes son el modelo triangular[1] y el trapezoidal[2].

Las normas subyacentes de WebRTC están siendo desarrolladas en conjunto por la *World Wide Web Consortium* (W3C) y la *Internet Engineering Task Force* (IETF).

1.1.1 ARQUITECTURA WEB

La semántica de la arquitectura web clásica está basada en el paradigma Cliente – Servidor, donde los navegadores realizan una petición HTTP (Protocolo de Transferencia de Hipertexto) al contenido en un servidor web, este contesta con una respuesta que contiene la información solicitada.

Los recursos proporcionados por el servidor están estrechamente asociados con la dirección URL (*Uniform Resource Locator*) o la URI (*Uniform Resource Identifier*).

1.2 WEBRTC [1]

1.2.1 INTRODUCCIÓN [2] [3]

WebRTC es considerada como una de las tecnologías más disruptivas en el actual mundo de las tecnologías de la información y comunicación. WebRTC forma parte de HTML5[3] y permite establecer comunicaciones en tiempo real sobre navegadores Web, sin necesidad de instalar extensiones no estándar o *plugins* propietarios. La iniciativa WebRTC es un proyecto apoyado por Google, Mozilla, Opera, entre otros.

---

[1] Modelo triangular: Escenario en el cual ambos navegadores ejecutan la misma aplicación web, desde un mismo servidor web.
[2] Modelo trapezoidal: Escenario en el cual ambos navegadores ejecutan una aplicación web descargada desde servidores web diferente.
[3] HTML5: *HyperText Markup Language* versión 5, quinta versión importante del lenguaje de marcado para la elaboración de páginas web.

WebRTC es un proyecto de fuente o código abierto, que habilita capacidades de comunicación en tiempo real en navegadores web, vía API[4] de JavaScript y permite que todos ellos se comuniquen a través de un conjunto común de protocolos.

WebRTC está pensado para comunicaciones *peer to peer* o P2P, en la que no se necesita un servidor intermediario, salvo para realizar la conexión inicial.

WebRTC ignora la parte de señalización, permitiendo a los fabricantes de sistemas VoIP[5] emplear WebRTC independientemente del protocolo de señalización usado para establecer la llamada.

Las opciones más populares para la señalización son: JSON (*JavaScript Object Notation*) sobre WebSockets y SIP[6] (*Session Initiation Protocol*) sobre WebSocket. Una vez realizada la conexión el servidor deja sus funciones, para tener una comunicación *peer to peer*.

1.2.2 HISTORIA

A comienzos del año 2010, Google finaliza la adquisición de On2[7], compañía desarrolladora de la serie de *codecs* VP, con su última versión, VP8. On2 planteó estos *codecs* como una patente libre en reemplazo a los *codecs* de series H.26X, los cuales se encuentran estandarizados, patentados y ampliamente usados.

Posteriormente, la compañía On2 realizó un *Open Source* de sus tecnologías de forma mundial y VP8 paso al nombre de WebM[8]. La idea era reemplazar H.264 para videos de la web, con ello, conllevar a una reducción de costos por patentes de todo el mundo.

Durante el año 2010, Google también adquiere GIPS[9] compañía conocida por su *framework* de comunicación, herramienta tecnológica que permite el desarrollo de

---

[4] API: Interfaz de programación de aplicaciones, conjunto de subrutinas, funciones y procedimientos para ser utilizado por otro *software*, como una capa de abstracción.
[5] VoIP: Voz sobre IP, es un conjunto de recursos que hacen posible que la señal de voz viaje a través de Internet empleando el protocolo IP.
[6] SIP (*Session Initiation Protocol*): Es un protocolo desarrollado por IETF con la intención de ser el estándar para la iniciación, modificación y finalización de sesiones multimedia.
[7] Originalmente conocida como *The Duck Corporation*, es una corporación especializada en desarrollar tecnologías de compresión de video.
[8] Formato multimedia abierto y libre desarrollado por Google y orientado a usarse con HTML5.
[9] Corporación desarrolladora de *software* de procesamiento de voz y video en tiempo real para redes IP.

las aplicaciones de VoIP. Google toma herramientas de GIPS [4], las cuales son publicadas en código abierto, lo que provoca que los *codecs* de voz y de videos patentados tomen menor relevancia.

El propósito fundamental tras esta concepción fue la de contar con tecnologías de procesamiento y de medios de codificación, en los medios de comunicación bidireccionales disponibles en los navegadores.

Las normas para WebRTC fueron presentadas y aprobadas, como un estándar ante la IETF y la W3C [5].

1.2.3 ARQUITECTURA WEBRTC

Los componentes dentro de WebRTC, tal y como se muestran en la figura 1.1, abarcan los *codecs*, motores de medios y capa de transporte.

Esto facilita el uso de comunicaciones en tiempo real por parte de los desarrolladores de aplicaciones web, ya que se reduce en gran parte las líneas de código necesarias para realizar tareas JavaScript, sin necesidad de ser expertos en estas tecnologías.

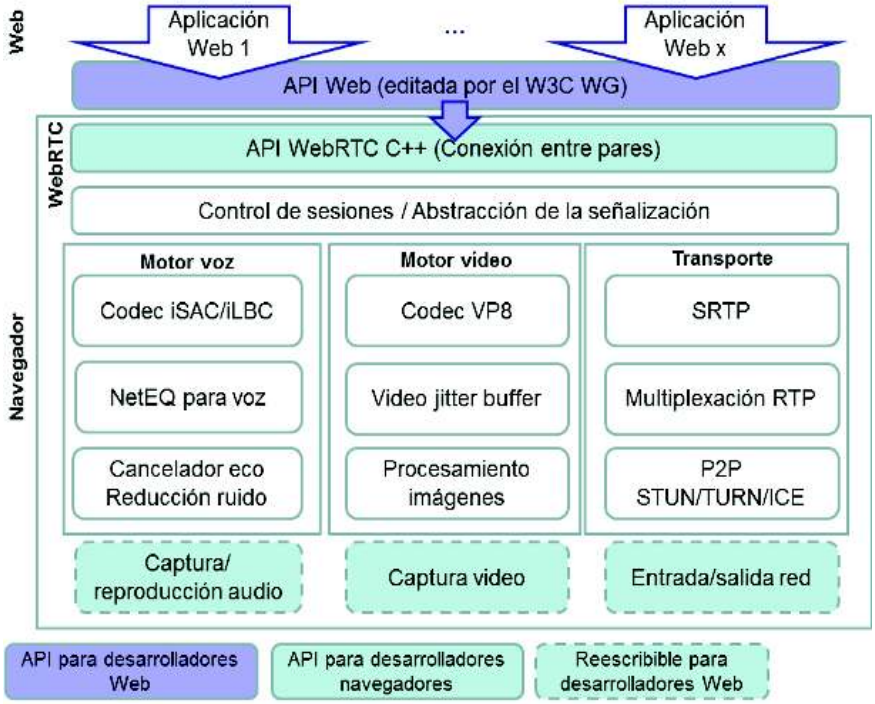


Figura 1.1 Arquitectura WebRTC [6]

WebRTC extiende la semántica cliente – servidor, mediante la introducción del paradigma de comunicación *peer-to-peer* entre navegadores. El modelo arquitectónico trapezoidal de WebRTC, utiliza el proceso de señalización SIP (*Session Initiation Protocol*), entre dos servidores web, el cual se indica en la figura 1.2:

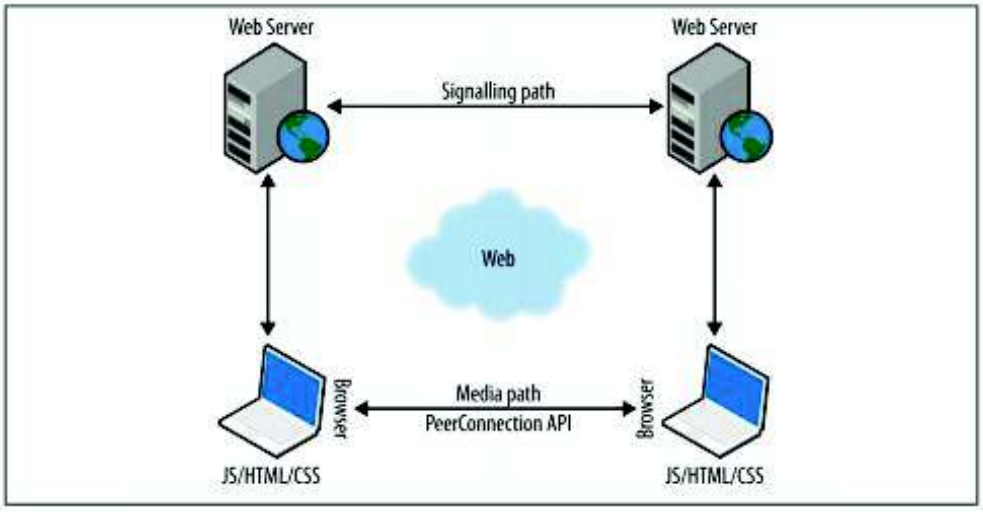


Figura 1.2: Modelo Trapezoidal WebRTC [1]

Dentro del modelo trapezoidal de WebRTC (fig. 1.2), los clientes ejecutan una aplicación web, que puede ser descargada desde diferentes servidores web.

El proceso de señalización es utilizado para administrar y terminar las sesiones de comunicación. Generalmente transportado por HTTP, WebSocket, SuperSocket o XSockets, a través de los servidores web que los pueden modificar, trasladar y administrar, cuando esto lo amerite.

Sin embargo, la señalización entre los navegadores y los servidores, no se ha estandarizado en WebRTC, como se mencionó en la sección 1.2.1, ya que es considerado parte de la aplicación.

En cuanto a la ruta de los datos, las API de WebRTC y en concreto la API *PeerConnection*, permite que los medios fluyan directamente entre los navegadores. Los dos servidores pueden comunicarse mediante protocolos de señalización como SIP, JSON o algún otro protocolo de señalización propietario desarrollado por el fabricante. El escenario más común y desarrollado en el presente trabajo, es aquel en que ambos navegadores ejecutan la misma aplicación

web, en la misma página web, este modelo se denomina triangular (Figura 1.3) y presenta muchos beneficios por su simplicidad.

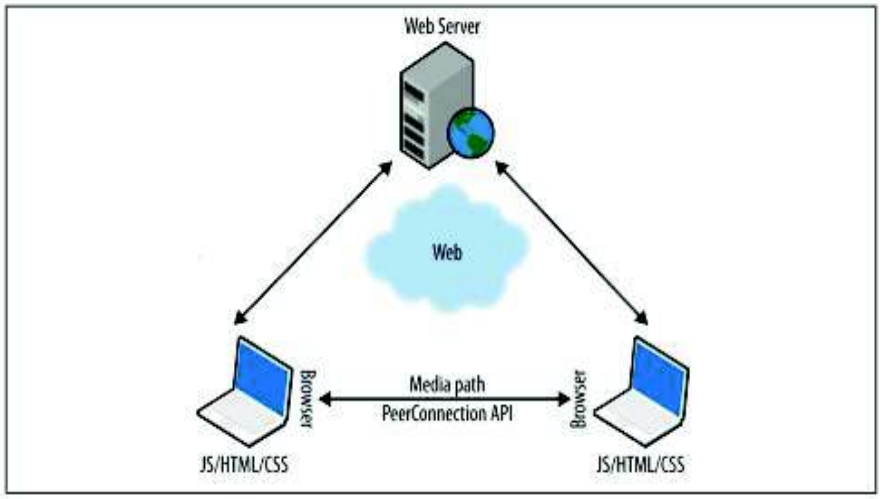


Figura 1.3: Modelo Triangular WebRTC [1]

1.2.4 WEBRTC EN NAVEGADORES

Una aplicación web WebRTC, típicamente desarrollada como una mezcla de HTML y JavaScript, interactúa con los navegadores a través de API WebRTC estandarizadas, permitiendo de esta manera controlar las funciones en tiempo real que se ejecutan en el navegador, (figura 1.4).

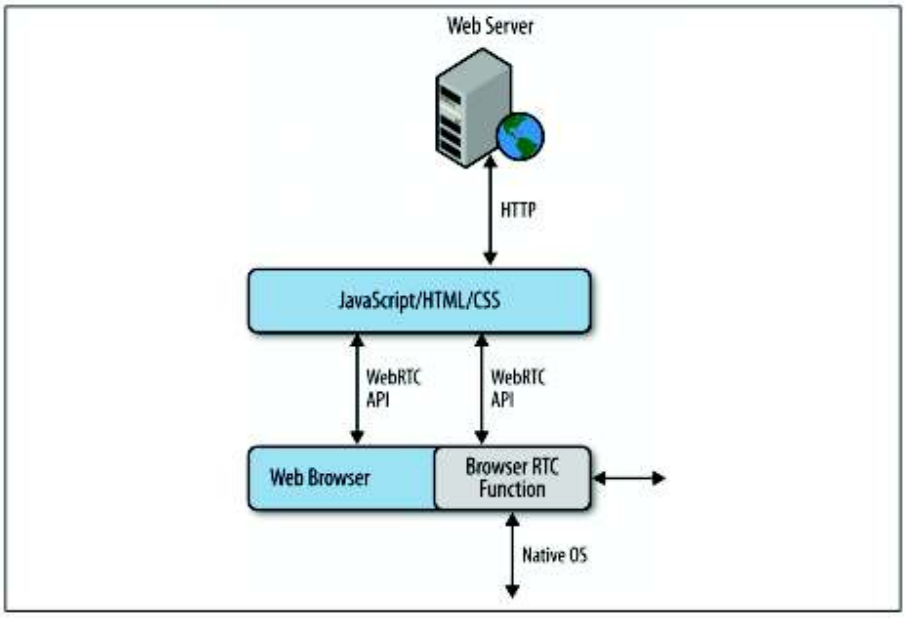


Figura 1.4: Comunicación en tiempo real en navegadores [1]

Las API WebRTC permiten un flujo continuo de los datos, que se transmiten a través de la red, a fin de permitir una comunicación directa entre los navegadores.

1.2.5 INTERFACES DE APLICACIÓN WEBRTC

El conjunto de interfaces se encuentran siendo desarrolladas por la *World Wide Web Consortium* (W3C), permite que las aplicaciones JavaScript aprovechen las capacidades en tiempo real de los navegadores. Los flujos de datos se cifran usando DTLS[10] 1.

Los *codecs* de voz seleccionados para WebRTC son el tradicional ITU-T G.711 y el reciente IETF RFC 6716 Opus, los cuales no necesitan licencia. Opus además de ser gratuito es de código abierto.

El *codec* de video más popular es H.264 pero presenta costos por licenciamiento asociados a patentes, por otro lado VP8 cuyo código fuente está disponible a la comunidad de *software* libre, es el *codec* donde se han realizado las primeras implementaciones sobre Chrome, aunque se espera que WebRTC termine soportando ambos *codecs*.

A continuación, en la tabla 1.1 se resumen de los protocolos usados en WebRTC

Tabla 1.1 Protocolos de WebRTC

Característica	**WebRTC**
Señalización	Sin definir
Medios	RTP/RTCP
Codecs de voz	G.711 y Opus
Codecs de video	VP8
Seguridad de los medios	SRTP

Las API que conforman WebRTC son las siguientes:

[10] DTLS: Basado en TLS (*Transport Layer Security*), se utiliza para el cifrado de las conexiones entre las aplicaciones que se comunican a través de UDP.

1.2.5.1 MediaStream

MediaStream es una representación abstracta de un flujo de datos, de audio y/o de video, ya que puede acceder a estos desde dispositivos locales como cámaras web y micrófonos.

LocalMediaStream representa un flujo de medios locales, capturados por un dispositivo local. Para crear y usar este flujo de medios locales, las aplicaciones web requieren acceso al usuario, a través de la función *getUserMedia().* La aplicación especificará el tipo de medio al cual esta requiere acceso.

El selector de dispositivos en la interfaz del navegador sirve como un mecanismo para conceder o denegar el acceso. Una vez que la aplicación se encuentra en marcha, puede revocar su propio acceso llamando a la función *stop()* en el *LocalMediaStream.*

El plano de señalización de los medios de comunicación de los extremos, se lleva a cabo conjuntamente con el uso de SRTP[11] (*Secure Real-Time Transport Protocol*), utilizado para transportar el flujo de medios, y RTCP[12] (*RTP Control Protocol*), utilizado para monitorear las estadísticas de la transmisión asociado al flujo de datos. DTLS es usado por SRTP y la gestión de la asociación.

1.2.5.2 PeerConnection

PeerConnection permite a dos usuarios comunicarse directamente, navegador a navegador. Ello representa una asociación con su par remoto, usualmente una instancia de la misma aplicación JavaScript ejecutándose en el extremo remoto.

Las comunicaciones se coordinan a través de un canal de señalización proporcionado por un código *scripting* en la página a través del servidor web.

Una vez que se establece una conexión entre pares, el flujo de medios puede ser enviado directamente hacia el navegador remoto. De ser necesario, *PeerConnection* utiliza el protocolo ICE (*Interactive Connection Establishment*)

---

[11] SRTP: Proporciona cifrado, autenticación del mensaje e integridad, y protección contra reenvío a los datos RTP en aplicaciones *unicast* y *multicast*.

[12] RTCP: Es un protocolo de comunicación que proporciona información de control que está asociado con un flujo de datos para una aplicación multimedia. Trabaja junto con otros protocolos en la parte de transporte.

junto con los servidores STUN y TURN, para permitir que los flujos basados en UDP, atraviesen las barreras del *Firewall* y el NAT[13].

ICE permite que los navegadores obtengan suficiente información acerca de la topología de la red, dado que se despliega a través de ella, para buscar la mejor ruta explotable.

El uso de ICE también proporciona una medida de seguridad, ya que impide que las aplicaciones y las páginas web no confiables, envíen datos a los *host* que no esperan recibir esta información.

1.2.5.3 DataChannel

La API *DataChannel* está diseñada para proporcionar un servicio de transporte genérico, permitiendo a los navegadores web el intercambio de datos de forma bidireccional *peer-to-peer*.

1.3 HTML5

HTML 5 no es una nueva versión del antiguo lenguaje de etiquetas, tampoco una mejora de ella, sino un nuevo concepto para la construcción de sitios y aplicaciones web, en una era que combina dispositivos móviles, computación en la nube y trabajos en red. [7]

HTML 5 provee básicamente tres características: estructura, estilo y funcionalidad. HTML 5 es considerado el producto de la combinación de HTML, CSS[14] y JavaScript, ya que, estas tecnologías son complementarias entre sí y actúan como una sola unidad organizada bajo la especificación de HTML5.

HTML está a cargo de la estructura, CSS presenta esa estructura y el contenido en pantalla, finalmente las API de WebRTC están desarrolladas en JavaScript, permitiendo a los desarrolladores Web integrar comunicaciones en tiempo real dentro de sus aplicaciones. Más allá de esta integración, la estructura sigue siendo parte esencial de un documento. La misma provee los elementos necesarios para

[13] NAT: Traducción de direcciones de Internet, convierte en tiempo real las direcciones IP utilizadas en los paquetes transportados.
[14] CSS: Hoja de estilo en cascada, lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML. La idea fundamental es la de separar la estructura de un documento de su presentación.

ubicar contenido estático o dinámico y es también una plataforma básica para aplicaciones.

1.3.1 ORGANIZACIÓN

La figura 1.5 representa un diseño común encontrado en la mayoría de los sitios web. En general se pueden identificar las siguientes secciones:

- *HEADER:* Es la cabecera de la página web, donde se encuentra necesariamente el título, donde se describe el contenido del sitio.
- *NAV:* Conocida como la barra de navegación, permite a los desarrolladores, desplegar información específica sobre el sitio o aplicación web, permitiendo encontrar los recursos del sistema más rápidamente a los usuarios.
- *SECTION:* Normalmente, es la sección más importante de la página, ya que es aquí donde se coloca todo el contenido del sitio, imágenes, videos, textos, etc.
- *ASIDE:* Contenido que está tangencialmente relacionado con el contenido que le rodea.
- *FOOTER:* Parte inferior de la estructura web, en la que generalmente se incluye *links* de navegación, enlaces de interés, *copyright* o botones a las redes sociales del sitio.

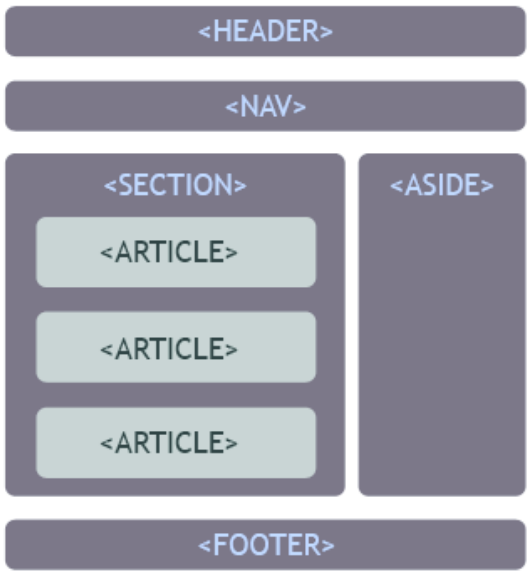


Figura 1.5: Esquema de las etiquetas para HTML5 [8]

1.3.2 LIBRERÍAS EXTERNAS DE HTML5

HTML 5 fue desarrollado para expandir la web utilizando un *set* de tecnologías estándar que todo navegador pueda entender y procesar. Fue creado para proveer todas las herramientas para sus posteriores desarrollos.

HTML5 fue conceptualizado para no depender de tecnologías de terceras partes. Antes de la aparición de HTML 5, varías librerías JavaScript fueron desarrolladas para superar las limitaciones de las tecnologías disponibles al momento. Algunas de estas librerías tenían propósitos específicos. Las siguientes librerías no son parte de HTML 5 pero son parte importante de la web y son actualmente usadas en sitios y aplicaciones web exitosas. Junto con el resto de las funciones incorporadas por esta especificación, mejoran al lenguaje JavaScript.

- *jQuery*

  Librería gratuita, diseñada para simplificar la creación de sitios web.
- *Google Maps*

  Complejo *set* de herramientas que permite desarrollar cualquier servicio de mapeado para la web.

1.4 JAVASCRIPT

Es un lenguaje interpretado orientado a objetos, que se utiliza en millones de páginas web y aplicaciones de servidor en todo el mundo. Es un lenguaje de programación dinámico que soporta construcción de objetos. La sintaxis básica es similar a Java y C++.

Las capacidades dinámicas de JavaScript incluyen construcción de objetos en tiempo de ejecución, listas, variables de parámetros, variables que pueden contener funciones, creación de *scripts* dinámicos y recuperación de código fuente. Algunas innovaciones permitieron cambiar la idea que se tenía de JavaScript gracias a los nuevos motores de interpretación con los que cuenta, creados para acelerar el procesamiento de código.

El éxito que han logrado algunos motores de interpretación, es transformar el código JavaScript en código de máquina para lograr velocidades de ejecución similares a las de aplicaciones de escritorio.

1.4.1 API HTML5 – JAVASCRIPT [9]

HTML 5 introduce varias API para proveer acceso a librerías robustas desde simple código JavaScript. Estas librerías permiten la realización de varias actividades, las más utilizadas se encuentran en la tabla 1.2:

Tabla 1.2 Librerías HTML5 – JavaScript

API	Descripción
Canvas	Provee una superficie de dibujo
Drag and Drop	Incorpora capacidad para arrastrar y soltar elementos
Geolocation	Establece la ubicación física del dispositivo usado
Storage	Almacenamiento de datos
File	API destinada a operar con archivos
Communication	XMLHttpRequest *Level* 2, *Cross Document Messaging*, Web Sockets.
Web Workers	Posibilidad de procesar código detrás de escena
History	Retorna a estados previos de la aplicación
Offline	Permite a la aplicación trabajar en línea o desconectado

1.5 XSOCKETS [10]

1.5.1 INTRODUCCIÓN

XSockets es un *framework* que permite una comunicación completa entre dos objetos en tiempo real. Se basa en una arquitectura modular, lo que permite una simplicidad en añadir (nuevas) o reemplazar (existentes) funcionalidades, actualmente se encuentra en la versión 5.0. Los módulos pueden modificarse o sustituirse dependiendo de las necesidades de cada usuario. XSockets proporciona una manera sencilla para trabajar con un subconjunto de clientes, es así como escenarios de comunicación compleja se los puede codificar con unas pocas líneas de código.

El marco es compatible con varios modelos diferentes de comunicación, tales como:

- Publicador / Suscriptor
- RPC / RMI[15]
- *DataSync*
- Mensajería simple

1.5.2 REQUERIMIENTOS

- **Requerimientos del Servidor**

El servidor puede ejecutarse en cualquier máquina que tenga .NET 4.5[16] o el equivalente en cada versión.

- **Requerimientos del Cliente**

Es difícil establecer los límites para el Cliente, ya que este se puede conectar con cualquier otro elemento, a través de TCP/IP. Por tal razón se muestran los clientes y las plataformas que en la actualidad se encuentra a disposición; sin embargo, cada usuario puede crear un cliente personalizado dependiendo de sus requerimientos o necesidades.
En la tabla 1.3, se muestran los clientes actualmente soportados por XSockets.

Tabla 1.3 Clientes actuales soportados por XSockets

Lenguaje	***Target***
C# .NET 4+	*Windows Desktop*
C# .NET 4.5	*Windows Phone/Store*
C# Xamarin	Android/iOS
JavaScript	*Browsers*
JavaScript (angularjs)	*Browsers*
JavaScript (nodejs)	*Everything*
C *for* Arduino	*MicroControllers*

[15] RPC / RMI: *Remote Procedure Call* y *Remote Method Incocation*, dos mecanismos que permiten al usuario invocar o llamar procesos que podrían ejecutarse en dos computadores diferentes.
[16] .NET 4.5: es una actualización en contexto compatible con versiones anteriores. El uso conjunto de .NET *Framework* 4.5 y el lenguaje de programación C#, permiten escribir aplicaciones Windows.

1.5.3 ARQUITECTURA

En la figura 1.6 se muestra una breve visión general de la composición de esta arquitectura.

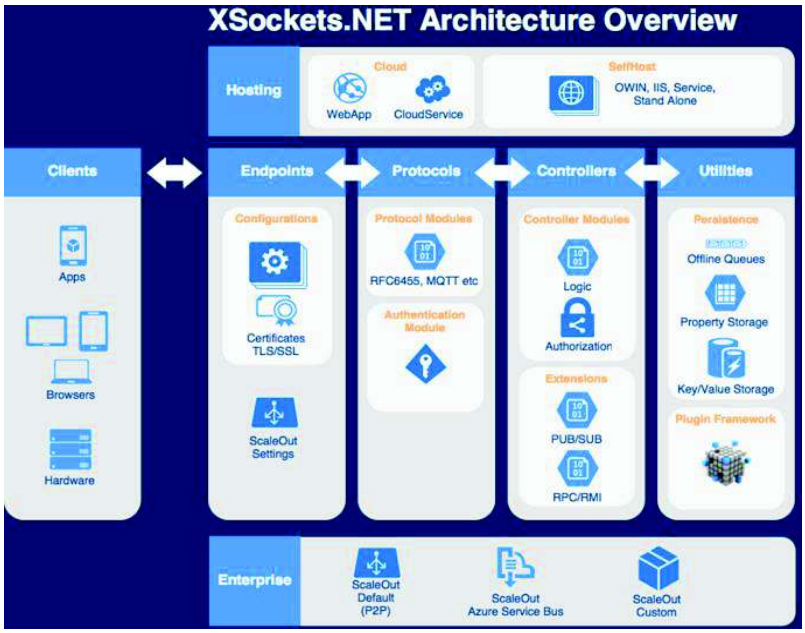


Figura 1.6: Arquitectura General de XSockets [10]

XSockets permite una *comunicación de protocolo cruzado* de manera que todos los clientes conectados puedan comunicarse con otros.

1.5.3.1 Clientes

Los clientes potenciales pueden ser cualquier objeto que tenga una conexión TCP/IP. Actualmente se tiene clientes para las siguientes plataformas, exceptuando aquellos clientes ya mencionados:

- C# .NET 4.5+
- IOS – Xamarin
- Android – Xamarin
- .NET *Micro Framework*
- Arduino
- *Photon*
- JavaScript
- NodeJS

1.5.3.2 EndPoints

Cada módulo de configuración añadido llegará a ser un punto extremo individual cuando inicia el servidor. Esto significa que los clientes pueden conectarse a los puntos finales separados. Esto es muy útil si se tiene diferentes niveles de seguridad en cada extremo.

1.5.3.3 Transporte

Si es necesario, se puede crear transportes personalizados y añadir el transporte a un extremo. Esto es muy usado, por ejemplo, cuando se realiza la conexión de objetos que no pueden usar certificados, pero aun así necesitan seguridad. Mediante la implementación de un transporte personalizado, se puede escoger su propio mecanismo de encriptación/desencriptación, tales como, encriptación simétrica, TLS – PSK, etc.

1.5.3.4 Protocolos

Los protocolos en XSockets son tratados como módulos, de esta forma se pueden añadir tantos protocolos como se desee. Por lo tanto los protocolos en XSockets requieren de algunos procedimientos:

- Comprobar si el protocolo de enlace enviado desde el cliente coincide con el protocolo.
- Convertir los datos de entrada hacia el formato interno *iMessage*.

Es una simple lógica que habilita *cross-protocol comunication*[17].

1.5.3.5 Utilidades

XSockets tiene una gran cantidad de características que reducen la complejidad en el desarrollo.

- *Offline queues*: Permite que los clientes *offline* obtengan los mensajes cuando estos se reconectan. Es decir, se tiene un cliente al cual se le han enviado varios mensajes, una vez reconectado los recupera.

[17] *Cross-protocol comunication*: Protocolo desarrollado por XSockets, permite la interoperabilidad de comunicación entre protocolos.

- Propiedades persistentes – Permiten mantener las propiedades del controlador, cuando un cliente se reconecta.
- Almacenamiento de clave/valor que permiten conservar los datos y obtenerlos más tarde.

Todas estas características son módulos y se pueden sobrescribir en sus propios módulos por defecto.

1.5.3.6 Hosting

XSockets se pueden alojar casi en cualquier lugar. Este no tiene ninguna dependencia con la versión del sistema operativo, con el servidor web (como IIS), etc.

Para la realización de las pruebas de funcionamiento y desarrollo del presente proyecto, el servidor funcionará de manera local (IIS *Express*).

1.5.4 CONFIGURACIÓN

1.5.4.1 Comunicación Cross EndPoint

Una característica muy importante en XSockets, es que se puede añadir tantas configuraciones como se desee o estas requieran.

Esto se puede ya que las configuraciones se basan en módulos recolectados en el arranque del servidor. Una vez que el servidor se ejecuta, permite la señalización entre los clientes de la aplicación, los dos *Peers* externos, una vez realizada la señalización, el servidor se desvincula. Este tipo de comunicación es muy utilizada en escenarios donde se tienen diferentes niveles de seguridad.

1.5.4.2 Configuración Automática

Si un servidor no está configurado, este utilizará una configuración automática, esto significa que el servidor creará puntos finales en el *localhost*, así como también todas las direcciones IP que se encuentran en la máquina.

El puerto configurado por defecto es el 4502, cuando el servidor se inicia se tiene lo mostrado en la figura 1.7:

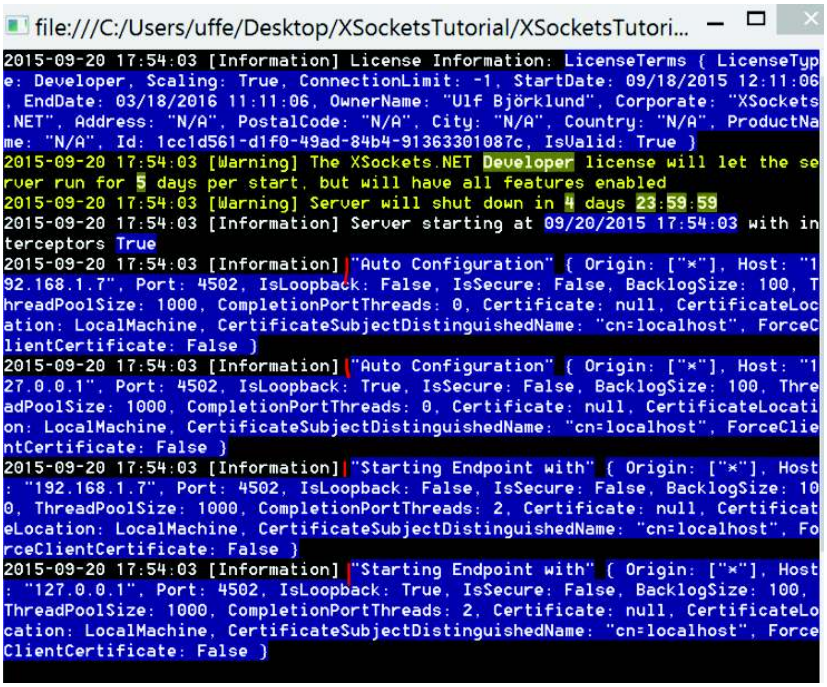


Figura 1.7: Configuración automática en XSockets

Los primeros dos marcadores muestran que el servidor identificó el *localhost* 127.0.0.1 y la dirección IP 192.168.1.7. Los siguientes marcadores muestran que el servidor creó los *endpoints* para cada IP identificada.

1.5.4.3 Configuración Inicial

Se puede configurar varios parámetros entre ellos se puede cambiar el puerto predeterminado. De igual forma se puede cambiar el punto final:

1.5.4.3.1 Punto Final

El objetivo principal de la configuración es comunicarle al servidor, en qué lugar y en qué puerto se desea que el *endpoint* sea creado.

1.5.4.3.2 Force Client Certificate

Si está configurado en el valor *true*, el servidor no aceptará las conexiones en los *endpoints*, a menos que el cliente pase un certificado de cliente válido. De igual forma habilita la obtención de información acerca de la certificación del cliente sobre la conexión.

1.5.4.3.3 Origen (Procedencia)

Por defecto el origen se establece en “*”. Lo que significa que cualquier persona puede conectarse independientemente de su origen. Si se desea realizar un filtro,

para permitir el tráfico de determinados orígenes, se puede especificar mediante una lista separada por comas.

1.5.5 CONTROLADORES

Los controladores son el corazón de la comunicación en XSockets. Los mensajes terminan en los controladores personalizados y los métodos en estos definidos.

Un controlador puede ser comparado con un controlador MVC (Modelo – Vista – Controlador). Pero existente una gran diferencia, ya que en XSockets se dispone de un estado en el controlador, esto significa que cada cliente tiene su propia instancia del controlador. Gracias al estado es muy simple apuntar a un subconjunto de clientes conectados en base al estado de cada cliente. Dependiendo de la solución se puede utilizar el controlador de una forma más clásica, donde se llama a una capa de servicio para realizar operaciones CRUD[18].

1.5.5.1 Creación de un Controlador

Para crear un controlador se debe heredar de la clase *XSocketController*. Esta clase se encuentra en *XSocket.Core.XSocket.* Un ejemplo se muestra en la figura 1.9.

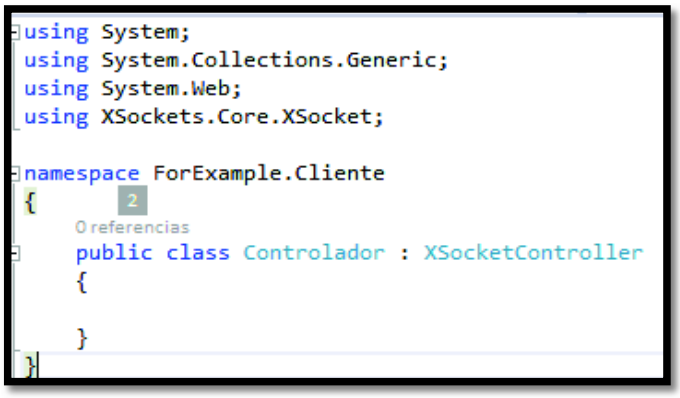


Figura 1.8: Ejemplo demostrativo donde se hereda la clase *XSocketController*

1.5.5.2 OnMessage

Un controlador básico se puede ver en la figura 1.10 en donde se destaca el método *OnMessage*.

[18] CRUD: Acrónimo de *Create, Read, Update* y *Delete*, se usa para referirse a las funciones básicas en base de datos de la capa de persistencia en un *software*.

```
using System.Collections.Generic;
using System.Web;
using XSockets.Core.XSocket;
using System.Threading.Tasks;
using XSockets.Core.Common.Socket.Event.Interface;
using XSockets.Core.XSocket.Helpers;

namespace ForExample.Cliente
{
    0 referencias
    public class Controlador : XSocketController
    {
        0 referencias
        public override Task OnMessage(IMessage message)
        {
            //Envía el mensaje a todos los clientes conectados a este controlador
            this.InvokeToAll(message);
        }

    }
```

Figura 1.9: Ejemplo demostrativo donde se utiliza el método *OnMessage*

La forma de trabajo del controlador es la siguiente: una vez que un mensaje llega al controlador, este buscará aquel método que coincida con el nombre del *tema*. Así que si hay un método llamado *SendMessage* y el *tema* se llama *Sendmessage* el *framework* invocará ese método.

En aquel caso que no exista el método coincidente con un tema en el controlador, se buscará el *override* (sobrecarga) del método *OnMessage.* Si el *override* no es encontrado, el mensaje no será procesado.

1.5.5.3 Controlador Genérico

Un controlador genérico puede ser muy útil, cuando se desea contar con un prototipo de forma ágil y rápida, sin necesidad de escribir código del lado del servidor, es decir, simplemente realizamos las configuraciones básicas en el lado del cliente.
En combinación con los API del cliente se puede realizar varias actividades avanzadas.

1.5.5.4 Métodos Personalizados

Ya se conoce como utilizar el método *OnMessage,* y como el *framework* decide invocar cada método. La figura 1.11 presenta un ejemplo para la invocación de métodos:

```
using System;
using System.Collections.Generic;
using System.Web;
using XSockets.Core.XSocket;
using System.Threading.Tasks;
using XSockets.Core.Common.Socket.Event.Interface;
using XSockets.Core.XSocket.Helpers;

namespace ForExample.Cliente
{
    0 referencias
    public class Controlador : XSocketController
    {
        0 referencias
        public override async Task OnMessage(IMessage message)
        {
            //Envía el mensaje a todos los clientes conectados a este controlador
            await this.InvokeToAll(message);
        }
```

Figura 1.10: Ejemplo demostrativo, forma de utilizar la invocación de métodos.

Los métodos implementados, solo responderán cuando se realice una comparación, cuando estos coincidan se realiza la implementación de cada línea de código que el método tiene definido.

1.5.5.5 Atributo Controller Event

Este atributo permite la sobrecarga de métodos, esto gracias a que se asignan otros nombres a los métodos, de tal forma que se permita un enrutamiento eficaz de los mensajes. Para hacer uso de tal atributo, se hereda de la clase *XSockets.Core.Common.Socket.Event.Attributes,* como se indica en el ejemplo de la figura 1.12:

```
using System.Collections.Generic;
using System.Web;
using XSockets.Core.XSocket;
using System.Threading.Tasks;
using XSockets.Core.Common.Socket.Event.Interface;
using XSockets.Core.XSocket.Helpers;
using XSockets.Core.Common.Socket.Event.Attributes;

namespace ForExample.Cliente
{
    0 referencias
    public class Controlador : XSocketController
    {
        0 referencias
        public override async Task OnMessage(IMessage message)
        {
            //Envía el mensaje a todos los clientes conectados a este controlador
            await this.InvokeToAll(message);
        }

        [ControllerEvent("F1")]
        0 referencias
        public async Task Sample()
        {
            //Envía el mensaje a todos los clientes con el tema "say"
            await this.InvokeToAll("Sample: Hola a todos", "Sample");
        }
```

Figura 1.11: Ejemplo demostrativo donde se muestra cómo utilizar los controladores.

Esto no solo es útil para la sobrecarga, también es un método útil para ahorrar *bytes* en la comunicación, facilita que a los métodos se les de nombres más cortos y conseguir una comunicación más eficiente.

1.5.5.6 Alias de Controladores/Métodos

Nombres largos significan más datos para pasar de ida y vuelta, entre servidor(es) y cliente(s). Para ello se puede dar a los controladores y métodos un alias que se utilizará en la comunicación.

Esto se lo realiza mediante la adición del atributo *XSocketMetadata,* ubicado en la clase *XSockets.Plugin.Framework.Attributes.*

A continuación, se presenta en la figura 1.13 un ejemplo del uso de este atributo.

```
using XSockets.Core.XSocket.Helpers;
using XSockets.Core.Common.Socket.Event.Attributes;
using XSockets.Plugin.Framework.Attributes;

namespace ForExample.Cliente
{
    [XSocketMetadata("C1")]
    0 referencias
    public class Controlador : XSocketController
    {
        0 referencias
        public override async Task OnMessage(IMessage message)
        {
            //Envía el mensaje a todos los clientes conectados a este controlador
            await this.InvokeToAll(message);
        }
```

Figura 1.12: Ejemplo demostrativo donde se muestra cómo utilizar el alias de un controlador.

1.5.6 SEGURIDAD

1.5.6.1 Autorización

El atributo *Authorize* se puede utilizar en los controladores ya sea a nivel de clase o de método. Esta forma de seguridad es muy utilizada cuando se tiene sistemas donde se implementa roles para los usuarios. También se puede anular dichos atributos si se desea desarrollar una lógica personalizada.

1.6 MODELO, VISTA, CONTROLADOR

1.6.1 INTRODUCCIÓN

Es un patrón de arquitectura de las aplicaciones *software*, separa la lógica de negocio de la interfaz de usuario, por lo que facilita la evolución por separado de ambos aspectos y también incrementa la reutilización de código y la flexibilidad de este. Se reduce el tiempo que se necesita para desarrollar aplicaciones con interfaces de usuario, ya que se puede dividir la complejidad en capas individuales y tratarlas de forma separada. [11] MVC es utilizado en múltiples plataformas (tales como las mostrados en la tabla 1.4), para el desarrollo de este proyecto se utiliza ASP .NET.

Tabla 1.4 *Framework*s utilizados en MVC

Framework	**Descripción**
XForms	Formato XML estándar del W3C para la especificación de un modelo de proceso de datos XML e interfaces de usuario como formularios web.
GTK+	Escrito en C, *toolkit* creado por Gnome para construir aplicaciones gráficas.
ASP.NET MVC *Framework*	Microsoft
Google Web *Toolkit*	GWT, para crear aplicaciones Ajax con Java.
Ruby on Rails	*Framework* para aplicaciones web con Ruby.

1.6.2 DEFINICIÓN DE LOS COMPONENTES DE MVC [12] [13]

- **Modelo**:

  Es el objeto que representa los datos del programa. Maneja los datos y controla todas sus transformaciones. El Modelo 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 cuando cambia el Modelo.

- **Vista**:
  Es el objeto que maneja la representación 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 de datos.
- **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.

1.6.3 FLUJO DE CONTROL DE MVC [14]

- El usuario realiza una acción en la interfaz.
- El controlador trata el evento de entrada.
- El controlador interpreta dicha acción.
- El controlador notifica al modelo la acción del usuario, lo que puede implicar un cambio del estado del modelo (si no se trata de una consulta)
- Se genera una nueva vista.
- La vista toma los datos del modelo.
- El modelo no tiene conocimiento directo de la vista.
- La interfaz de usuario espera otra interacción del usuario, que comenzará otro nuevo ciclo.

El controlador interpreta la acción que realiza un usuario, estos eventos puede ser dar *click* en un botón.

En la figura 1.14 se ilustra los elementos del patrón y la interrelación entre estos.

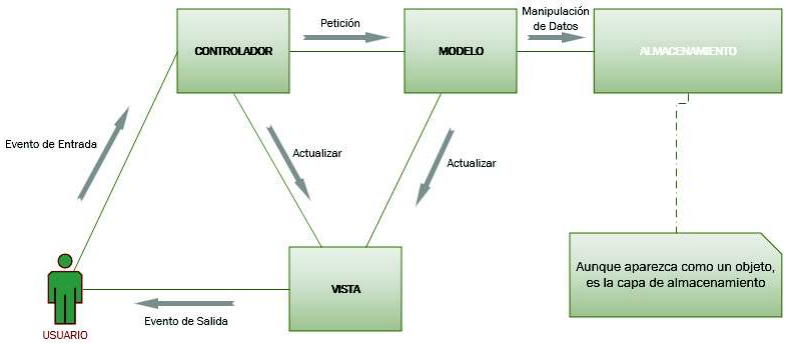


Figura 1.13: Interrelación entre los elementos del patrón MVC

Un modelo puede tener diversas vistas, cada una con su correspondiente controlador, aunque es posible definir múltiples vistas para un único controlador. El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página; el modelo es el Sistema de Gestión de la Base de Datos y la lógica de negocio; y el controlador es el responsable de recibir los eventos de entrada desde la vista. Una de las dificultades con las que se debe lidiar es la implementación del patrón es el hecho de que es posible incorporar en las clases de la vista gran parte o todo el procesamiento de eventos. Con lo que el controlador puede quedar semi-oculto dentro de la vista.

1.6.4 MVC EN APLICACIONES WEB

- Vista: página HTML
- Controlador: genera el contenido HTML
- Modelo: información almacenada en una base de datos o en XML. [15]

Figura 1.14: Interrelación del patrón MVC en aplicaciones

1.6.4.1 Diferencias entre MVC y WebForms [16]

WebForms fue el primer modelo de programación de ASP .NET, que proporciona un gran nivel de abstracción, con un modelo de programación familiar basado en eventos y controles. En el presente proyecto se elige MVC, ya que es un modelo mucho más sencillo, a diferencia de *WebForms* que es un modelo de desarrollo de aplicaciones de escritorio adaptado a la web.

En general, la creación de una aplicación *WebForms* requiere menos esfuerzos de programación, que crear la misma aplicación utilizando el marco ASP.NET MVC. Sin embargo, los formularios web no permiten un desarrollo rápido de aplicaciones.

Por ejemplo, no siempre es posible especificar exactamente qué margen de beneficio podría ser generado por un control.

El marco de formularios *WebForms*, no se presta tan fácilmente como ASP.NET MVC para prácticas de desarrollo de patrones, tal como el desarrollo basado en pruebas.

1.6.4.2 Plantilla MVC

ASP.NET MVC fue diseñado para facilitar las prácticas de desarrollo basado en patrones, tales como desarrollo basado en pruebas y la inversión de control. El marco fomenta la separación de la capa de lógica de negocio de una aplicación web de su capa de presentación. Al dividir la aplicación en los modelos (M), puntos de vista (V), y los controladores (C), ASP.NET MVC puede hacer que sea más fácil de gestionar la complejidad en las aplicaciones más grandes. Con ASP.NET MVC, se trabaja de forma más directa con HTML y HTTP que en *Web Forms*. Por ejemplo, los formularios Web pueden preservar de forma automática el estado entre las peticiones HTTP.

La ventaja del modelo MVC es que permite tomar el control completo sobre lo que su aplicación exactamente está haciendo. La desventaja es que se tiene que escribir más líneas de código.

MVC fue diseñado para ser extensible, proporcionando a los desarrolladores la posibilidad de personalizar el marco para las necesidades de la aplicación. Además,

el código fuente de ASP.NET MVC se encuentra disponibles bajo una licencia de *software* libre. La plantilla MVC crea una muestra MVC 5, aplicación que utiliza *Bootstrap* para proporcionar características de diseño y tematización de respuesta. [17]

2 CAPÍTULO 2: DISEÑO DEL PROTOTIPO

En este capítulo se establecen los requerimientos de la aplicación, de tal forma que permita el desarrollo de cada módulo que lo integra, también se detalla los diferentes tipos de roles y permisos que se asignarán a los usuarios.

Se describen los diferentes tipos de diagramas con los cuales se desarrolló la aplicación.

La metodología a utilizar en el desarrollo de la aplicación es XP (*eXtreme Programming*), ya que permite una programación sumamente organizada y facilita los cambios.

2.1 ANÁLISIS DE REQUERIMIENTOS

Para realizar la fase de análisis de requerimientos y obtener información precisa para un correcto desarrollo de la aplicación, se ha acudido a una empresa dedicada al sector de las telecomunicaciones.

Se propuso el desarrollo de una aplicación web que permita realizar comunicación en tiempo real entre sus colaboradores.

Dentro de las características principales mencionadas se encuentran las siguientes funcionalidades:

- Permitir el intercambio de mensajes.
- Permitir el intercambio de archivos.
- Permitir video llamadas a través del navegador.
- Vistas de administración para acciones CRUD de usuarios y grupos

En esta sección se presenta la definición de roles, las acciones que realizan los usuarios, la priorización y la definición de los módulos de la aplicación en base a las historias de usuario.

2.1.1 USUARIOS, ROLES Y PERMISOS DE LA APLICACIÓN

Se ha definido dos actores que interactúan con la aplicación web, detallados a continuación:

- Administrador
  El administrador es el actor encargado de gestionar la aplicación web, realiza las acciones de CRUD de los usuarios y grupos; es decir, administra las cuentas de los usuarios.

Adicionalmente, podrá obtener reportes de los usuarios registrados en la aplicación web.

- Usuario
  Es aquel actor que utiliza y accede a las funcionalidades de comunicación en tiempo real, además podrá visualizar información detallada de todos los usuarios registrados en la aplicación web, este actor utilizará las funciones de comunicación en tiempo real.

Se establecen en general dos niveles de permisos en la aplicación, que están relacionados con los roles establecidos en esta, a continuación se detallan dichos permisos:

- Administrador:
  Cuando una persona inicia sesión como administrador, se le proporcionan todos los permisos en el sistema, de manera que se le proporcionan opciones adicionales para la gestión de los recursos de este, acorde con las funciones detalladas.
- Usuario:
  Cuando una persona inicia sesión como usuario, se limitan los permisos en el sistema, de forma que el usuario solo tiene ciertas opciones disponibles para la manipulación de los recursos, acorde con las funciones detalladas.

2.1.2 HISTORIAS DE USUARIO

Las historias de usuario son representaciones de requisitos de *software* escritas en lenguaje común de un usuario; principalmente utilizadas en metodologías de desarrollo ágil.

2.1.2.1 Formato de las Historias de Usuario

El formato de las historias de usuarios, se muestra en la tabla 2.1

Tabla 2.1 Formato de la Historia de Usuario

<table>
<tr><td>Número de Historia de Usuario</td><td colspan="2"></td><td>Fecha</td><td colspan="2"></td></tr>
<tr><td>Nombre de la Historia de Usuario</td><td colspan="5"></td></tr>
<tr><td rowspan="2">Tipo de Actividad</td><td>Nueva</td><td>Cambio</td><td>Afinamiento</td><td rowspan="2">Estimación</td><td rowspan="2"></td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td rowspan="2">Prioridad</td><td>Alta</td><td>Media</td><td>Baja</td><td rowspan="2">Iteración</td><td rowspan="2"></td></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>Programador</td><td colspan="5"></td></tr>
<tr><td>Descripción</td><td colspan="5"></td></tr>
</table>

- **Número de Historia de Usuario**

  El número de la historia de usuario, representa la identificación de dicha historia, generalmente se la representa con las letras “HU” y el número correspondiente.

- **Fecha**

  Fecha en la cual la historia de usuario fue escrita.

- **Nombre de la Historia de Usuario**

  Denominación asignada a una historia de usuario.

- **Tipo de Actividad**

  El tipo de actividad especifica si se trata de una actividad nueva, si se debe realizar un cambio o si esta debe afinarse.

- **Estimación**

  Medida del tiempo aproximado para la codificación de la historia de usuario, generalmente en horas.

- **Prioridad**

Prioridad asignada a la historia de usuario por el programador.

- **Descripción**

Información de los requerimientos establecidos en la fase del análisis de requerimientos por el cliente.

- **Programador**

Responsable del desarrollo de la historia de usuario.

2.1.3 DETALLE DE LAS HISTORIAS DE USUARIO

2.1.3.1 Módulo de Administración de Usuarios

La historia de usuario para la administración de usuarios se muestra en la tabla 2.2:

Tabla 2.2 HU1 - Administración de Usuarios

Número de Historia de Usuario	**HU1**		**Fecha**	**15/01/2016**	
Nombre de la Historia de Usuario	Administración de usuarios				
Tipo de Actividad	**Nueva**	**Cambio**	**Afinamiento**	**Estimación**	**20**
	X				
Prioridad	**Alta**	**Media**	**Baja**	**Iteración**	**1**
	X				
Programador	Carlos Mendoza				
Descripción	El administrador podrá realizar acciones de CRUD de los usuarios de la aplicación				

La historia de usuario para la administración de administradores se muestra en la tabla 2.3 que permite realizar el CRUD de aquellos usuarios encargados de la gestión:

Tabla 2.3 HU 2 - Administración de Administradores

Número de Historia de Usuario	**HU2**		**Fecha**	**16/01/2016**	
Nombre de la Historia de Usuario	Administración de administradores				
Tipo de Actividad	**Nueva**	**Cambio**	**Afinamiento**	**Estimación**	**30**
	X				
Prioridad	**Alta**	**Media**	**Baja**	**Iteración**	**1**
	X				
Descripción	El administrador podrá realizar acciones de CRUD de los administradores de la aplicación				

2.1.3.2 Módulo de Administración de Grupos

La historia de usuario para la gestión de grupos se muestra en la tabla 2.4:

Tabla 2.4 HU 3 - Administración de Grupos

Número de Historia de Usuario	**HU3**		**Fecha**	**17/01/2016**	
Nombre de la Historia de Usuario	Administración de grupos				
Tipo de Actividad	**Nueva**	**Cambio**	**Afinamiento**	**Estimación**	**10**
	X				
Prioridad	**Alta**	**Media**	**Baja**	**Iteración**	**1**
	X				
Programador	Carlos Mendoza				
Descripción	El administrador podrá crear y eliminar grupos de usuarios.				

La historia de usuario para la administración de miembros del grupo se puede observar en la tabla 2.5:

Tabla 2.5 HU 4 - Administración de miembros

Número de Historia de Usuario	**HU4**		**Fecha**	**18/01/2016**	
Nombre de la Historia de Usuario	Administración de miembros del grupo				
Tipo de Actividad	**Nueva**	**Cambio**	**Afinamiento**	**Estimación**	**15**
	X				
Prioridad	**Alta**	**Media**	**Baja**	**Iteración**	**1**
	X				
Descripción	El administrador podrá añadir y eliminar miembros del grupo.				

2.1.3.3 Módulo de Autenticación

La historia de usuario para la autenticación de usuarios se muestra en la tabla 2.6:

Tabla 2.6 HU 5 - Autenticación

Número de Historia de Usuario	**HU5**		**Fecha**	**19/01/2016**	
Nombre de la Historia de Usuario	Autenticación				
Tipo de Actividad	**Nueva**	**Cambio**	**Afinamiento**	**Estimación**	**15**
	X				
Prioridad	**Alta**	**Media**	**Baja**	**Iteración**	**2**
		X			
Descripción	Se validará el ingreso a la aplicación de usuarios y administradores.				

2.1.3.4 Módulo de Comunicación

La historia de usuario que describe las video-llamadas, se presenta en la tabla 2.7:

Tabla 2.7 HU 6 - VideoLlamadas

Número de Historia de Usuario	**HU6**		**Fecha**	**23/01/2016**	
Nombre de la Historia de Usuario	Video-llamadas				
Tipo de Actividad	**Nueva**	**Cambio**	**Afinamiento**	**Estimación**	**55**
	X				
Prioridad	**Alta**	**Media**	**Baja**	**Iteración**	**1**
	X				
Descripción	El usuario o el administrador podrán realizar video-llamadas con otro usuario o administrador.				

La historia de usuario para el intercambio de mensajes, se muestra en tabla 2.8:

Tabla 2.8 HU 7 - Intercambio de mensajes

Número de Historia de Usuario	**HU7**		**Fecha**	**24/01/2016**	
Nombre de la Historia de Usuario	Intercambio de mensajes				
Tipo de Actividad	**Nueva**	**Cambio**	**Afinamiento**	**Estimación**	**45**
	X				
Prioridad	**Alta**	**Media**	**Baja**	**Iteración**	**2**
		X			
Descripción	El usuario o el administrador podrán intercambiar mensajes con otro usuario o administrador o usuario registrado.				

La historia de usuario para el intercambio de archivos, se presenta en tabla 2.9:

Tabla 2.9 HU 8 - Intercambio de archivos

Número de Historia de Usuario	**HU8**		**Fecha**	**25/01/2016**	
Nombre de la Historia de Usuario	Intercambio de archivos				
Tipo de Actividad	**Nueva**	**Cambio**	**Afinamiento**	**Estimación**	**45**
	X				
Prioridad	**Alta**	**Media**	**Baja**	**Iteración**	**2**
		X			
Descripción	El usuario o el administrador podrán intercambiar archivos con otro usuario o administrador.				

2.1.3.5 HU Complementarias

Las historia de usuario, complementarias que presentan la información de la aplicación se muestran en la tabla 2.10:

Tabla 2.10 HU 9 - Información de la Aplicación

Número de Historia de Usuario	**HU9**		**Fecha**	**20/01/2016**	
Nombre de la Historia de Usuario	Información de la aplicación				
Tipo de Actividad	**Nueva**	**Cambio**	**Afinamiento**	**Estimación**	**12**
	X				
Prioridad	**Alta**	**Media**	**Baja**	**Iteración**	**3**
			X		
Descripción	El usuario y el administrador visualizarán información acerca de la aplicación.				

Información de contacto se presenta en la tabla 2.11:

Tabla 2.11 HU 10 - Información Programador

Número de Historia de Usuario	**HU10**		**Fecha**	**21/01/2016**	
Nombre de la Historia de Usuario	Información de contacto				
Tipo de Actividad	**Nueva**	**Cambio**	**Afinamiento**	**Estimación**	**10**
	X				
Prioridad	**Alta**	**Media**	**Baja**	**Iteración**	**3**
			X		
Programador	Carlos Mendoza				
Descripción	El usuario y el administrador visualizarán información acerca del desarrollador de la aplicación, con número de contacto.				

2.1.4 REQUERIMIENTOS DE FUNCIONALIDAD

A los requerimientos de la aplicación se los ha dividido en funcionales y no funcionales, en base a las historias de usuarios detalladas en la anterior sección. Los requerimientos funcionales se encuentran dentro de los objetivos planteados y los no funcionales, por elementos externos a ellos.

Dentro de los requerimientos funcionales se detallan los servicios que provee la aplicación, de tal manera que esta reaccione a entradas particulares, para cumplir con los requerimientos realizados en la fase de análisis de requerimientos, de tal forma que se cumpla satisfactoriamente los objetivos planteados.

Se detalla en total doce requerimientos funcionales y tres requerimientos no funcionales.

Los requerimientos no funcionales y funcionales se presentan a continuación en la tabla 2.12a y la tabla 2.12b:

Tabla 2.12a Requerimientos de la Aplicación

REQUERIMIENTOS FUNCIONALES	
RF1	La aplicación permitirá al administrador crear, modificar y eliminar usuarios.
RF3	La aplicación permitirá al administrador y al usuario modificar su información personal
RF4	La aplicación permitirá al administrador cambiar el rol de un usuario a rol de administrador y viceversa.
RF5	La aplicación permitirá al administrador crear, editar y eliminar grupos de usuarios registrados.
RF6	La aplicación permitirá al administrador añadir y eliminar miembros de un grupo.
RF7	La aplicación permitirá a una persona auto registrarse, proporcionando información personal asignando el rol de usuario.
RF8	La aplicación permitirá a un usuario iniciar sesión, ingresando correo electrónico y contraseña.
RF9	La aplicación proporcionará información acerca de su uso, tanto de cómo iniciar sesión, como del proceso de registro.
RF10	La aplicación permitirá a un usuario establecer una videollamada con otro usuario registrado.
RF11	La aplicación permitirá a un usuario enviar mensajes a otro usuario registrado.
RF12	La aplicación permitirá a un usuario transferir archivos de tamaño y tipos permitidos a otro usuario registrado.

Tabla 2.13b Requerimientos de la Aplicación **REQUERIMIENTOS NO FUNCIONALES**	
RNF1	Las contraseñas serán cifradas en la base de datos.
RNF2	Uso de contraseñas para acceder a la aplicación.
RNF3	La aplicación será adaptable al tamaño de la ventana del navegador.

2.1.5 PROPUESTA DE SOLUCIÓN

En la figura 2.1 se presenta la solución propuesta. El *Web Server* (Ítem 1 en la Figura 2.1) tiene la función de un servidor de señalización, que permite establecer una sesión entre los *Peers* (Navegadores), una vez que ya se ha obtenido la información necesaria para establecer la comunicación, el servidor de señalización se desvinculará. El módulo de comunicación (Ítem 2 en la Figura 2.1) es donde se alojan las API de WebRTC que permite a las aplicaciones del navegador realizar *chat* de video y uso compartido de archivos P2P sin la necesidad de instalar *plugins.*

Los módulos de administración (Ítem 3 en la Figura 2.1) y autenticación (Ítem 4 en la Figura 2.1) interactúan directamente con la base de datos, ya que aquí es donde se almacenarán los perfiles de los usuarios registrados en la aplicación y se validará su ingreso a esta. El módulo de información (Ítem 5 en la Figura 2.1) permitirá informar a los usuarios acerca de la manipulación de los recursos de la aplicación. Es posible establecer sesiones entre más de dos *Peers*, pero el desarrollo de este proyecto se centra en la comunicación de dos *Peers*. En base a las historias de usuarios detalladas en la sección anterior, se definen los siguientes módulos que conformarán la aplicación:

- Módulo de administración de usuarios
- Módulo de administración de grupos
- Módulo de autenticación
- Módulo de comunicación

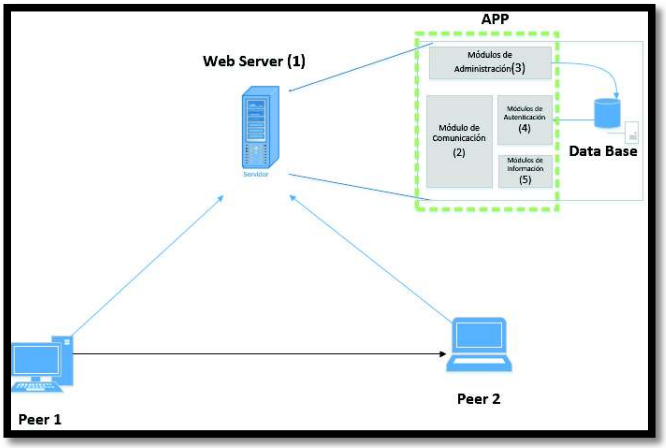


Figura 2.1 Solución Propuesta

2.1.6 HISTORIAS DE USUARIO POR ITERACIÓN

A continuación se detallará la planificación de entregables para el desarrollo de la aplicación web, basado en la estimación y la prioridad asignada a la historia de usuario.

2.1.6.1 Primera Iteración

Para la primera iteración se realizan los módulos de administración y un módulo de comunicación mostrados en la tabla 2.13:

Tabla 2.14 HU - Primera Iteración

Módulo	Historias de Usuario	Prioridad[19]	Estimación (horas)
Administración de Usuarios	Administración de usuarios	1	20
	Administración de administradores	1	30
Administración de Grupos	Administración de grupos	1	10
	Administración de miembros del grupo	1	15
Comunicación	Video-Llamadas	1	55
		Total	130

[19] Prioridad: Definición de valores de prioridad, Alta = 1, Media = 2, Baja = 3

2.1.6.2 Segunda Iteración

Para la segunda iteración se realizan el módulo de autenticación que permite identificar a los usuarios de la aplicación y la finalización del módulo de comunicación mostrado en la tabla 2.14:

Tabla 2.15 HU - Segunda Iteración

Módulo	Historias de Usuario	Prioridad	Estimación (horas)
Autenticación	Autenticación	2	15
Comunicación	Intercambio de mensajes	2	45
	Intercambio de archivos	2	45
		Total	105

2.1.6.3 Tercera Iteración

En la tercera iteración se presenta la información concerniente al uso de la aplicación, que sirve de ayuda para la manipulación de los recursos de esta, además también se puede visualizar información acerca del desarrollador.

En la tabla 2-16 se observa la tercera iteración:

Tabla 2.16 HU - Tercera Iteración

Historias de Usuario	Prioridad	Estimación (horas)
Información de la aplicación	3	12
Información de contacto	3	10
	Total	22

2.2 DESARROLLO DE LA APLICACIÓN

2.2.1 DISEÑO DE LA APLICACIÓN

En esta sección se determina el funcionamiento de la aplicación y se describen los diferentes módulos que lo integran, a través de los diferentes tipos de diagramas y procesos.

También se incluyen el diagrama de clases, que permite conocer algunos modelos que intervienen en el patrón MVC (modelo, vista, controlador), y también el diagrama entidad-relación, para la base de datos, que almacena la información de cada uno de los perfiles de los usuarios.

Luego se presentan los diagramas de procesos para cada módulo, estos diagramas permiten conocer cómo se realizan las consultas en la base de datos de la aplicación.

2.2.1.1 Diseño de la Arquitectura Propuesta

En base al análisis de requerimientos realizada en el apartado 2.2, se han definido los módulos detallados en la figura 2.1, (módulo de administración de usuarios, módulo de administración de grupos, módulo de autenticación y módulo de comunicación), que se muestra a continuación:

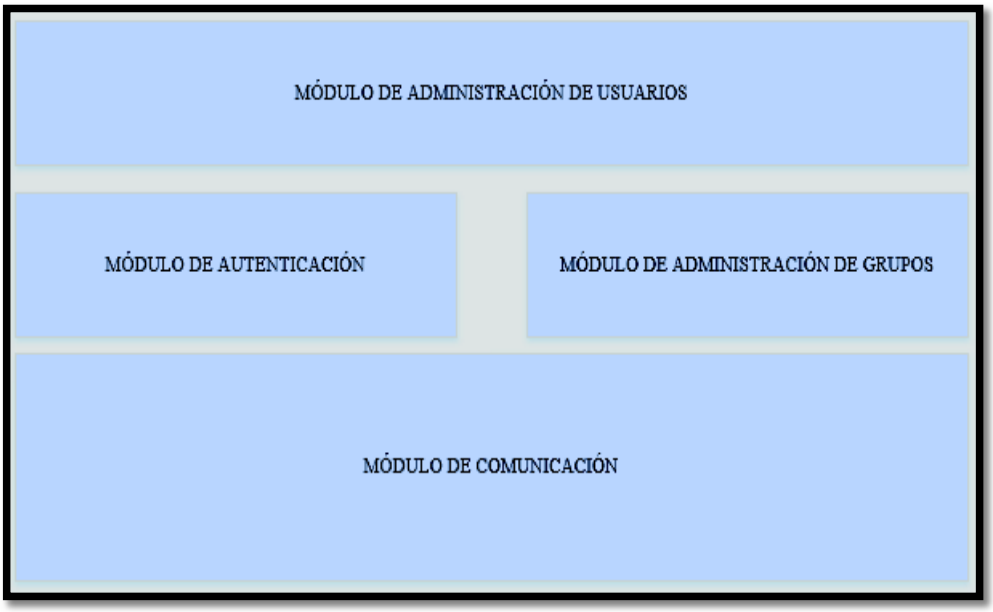


Figura 2.2 Arquitectura de la Aplicación Propuesta

2.2.1.2 Diagrama de Clases

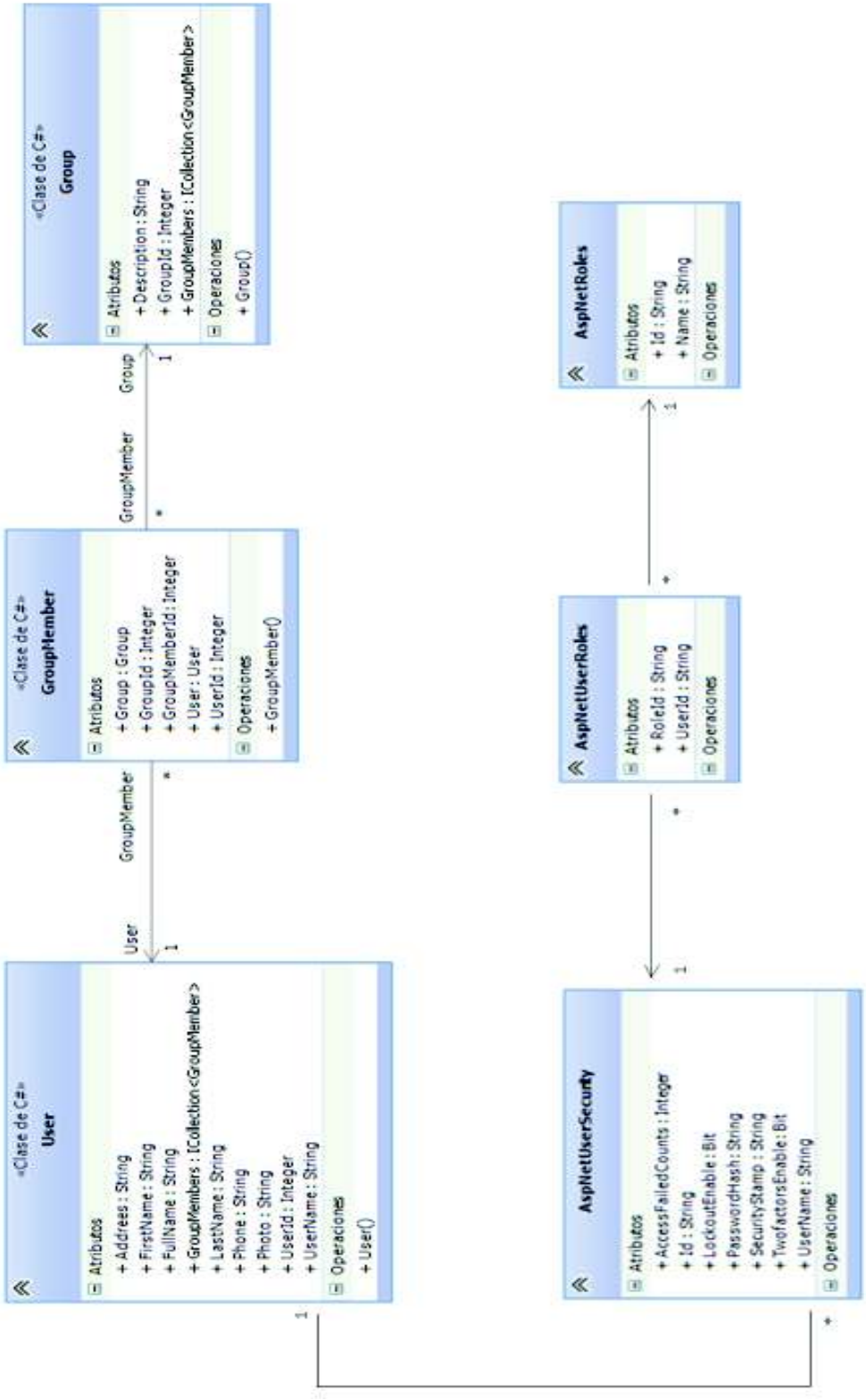


Figura 2.3 Diagrama de Clases

2.2.1.3 Diagrama de la Base de Datos

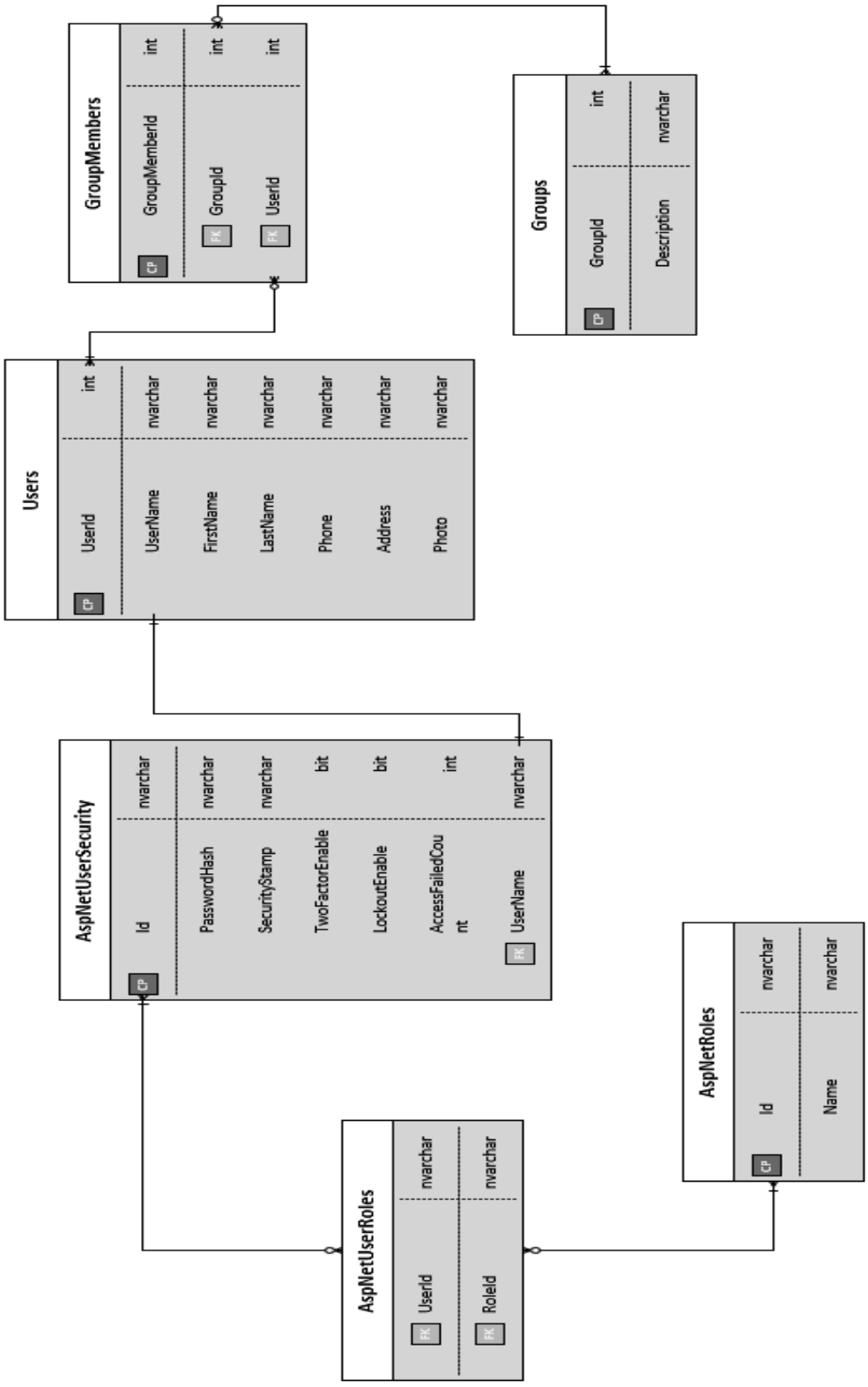


Figura 2.4 Diagrama de la Base de Datos

Al utilizar ASP .NET, se trabajó con el *framework* denominado *Entity Framework*[20], que permite acceder a los datos y crear las tablas del sistema. [18].

- **Tablas generadas con la creación del proyecto**

Las tablas *AspNetUserSecurity*, *AspNetUserRoles* y *AspNetRoles*, son generadas automáticamente y proporcionan seguridad al sistema, (las tablas no son desarrolladas por el programador), estas tablas son creadas internamente mediante un espacio de nombres denominado *System.Web.Security*[21] [19], lo que significa que se delega al *Framework* .NET la creación de roles y la creación de parámetros de seguridad para los usuarios.

La tabla *AspNetUserRoles* divide la relación muchos a muchos entre las tablas *AspNetUserSecurity* y *AspNetRoles*, que debido a las reglas de integridad referencial[22] no se puede tener.

También se han diseñado estas tablas con una relación de muchos a muchos, debido a que un rol puede tener muchos usuarios (varios administradores o varios usuarios), y un usuario puede tener muchos roles (para este sistema dos), es decir, una persona puede ser administrador y usuario a la vez, de esta manera ambos puedan hacer uso del módulo de comunicación.

- **Tablas desarrolladas**

Las tablas desarrolladas son *Users*, *GroupMembers* y *Groups*,

Existe una relación entre las tablas *Users* y *AspNetUserSecurity*, que permite proporcionar algunas opciones de seguridad en la aplicación; como por ejemplo, el cifrado de las contraseñas del usuario. La tabla *Users* contiene los datos que se ha propuesto en la fase de análisis de requerimientos, estos son, nombre, apellido, correo, teléfono, dirección y foto; almacenados en los campos *FirstName*, *LastName*, *UserName*, *Phone*, *Address* y *Photo* respectivamente.

---

[20] *Entity Framework*: Es un asignador objeto – relacional que permite a los desarrolladores de .NET trabajar con datos relacionales usando objetos específicos del dominio.
[21] *System.Web.Security*: Espacio de nombres que contiene las clases que se utilizan para implementar seguridad en aplicaciones ASP .NET.
[22] Regla de Integridad Referencial: Determina que todos los valores que toma una clave foránea deben ser valores nulos o valores que existen en la clave primaria que referencia.

Finalmente, se tienen dos tablas llamadas *Groups* y *GroupMembers* que se han integrado con la finalidad de ordenar de mejor manera a los usuarios y administradores.

Es decir, el administrador puede gestionar y agrupar a usuarios por área de trabajo o funciones que desempeña, algunos ejemplos son: Contabilidad, Administración, Técnica, etc.

2.2.1.4 Diseño de Procesos y Vistas por Módulo

La aplicación se describe como un conjunto de procesos que se llevan a cabo dentro de esta.

Se definió los procesos de cada uno de los módulos exceptuando aquellos que poseen tareas similares.

Para la creación de las vistas que tiene la aplicación, se utilizó el *software* de código abierto *Pencil Project*[23] que permite crear diagramas y realizar interfaces de usuario.

Las vistas que se detallan a continuación han sido diseñadas a partir del análisis de requerimientos y los diferentes tipos de diagramas obtenidos en el diseño de la aplicación.

2.2.1.4.1 Módulo de Autenticación

- **Vista para el inicio de sesión**

Esta vista permite a las personas poder autenticarse en la aplicación, para ello deben ingresar sus credenciales (correo electrónico y contraseña).

Esta opción es válida para aquellas personas que ya se registraron, es decir, ya son usuarios de este, o el administrador ya creo un perfil del usuario solicitado en la aplicación. La aplicación valida si los campos no se encuentran vacíos, que el campo para ingresar el correo electrónico sea válido, etc. Se notifica al usuario con mensajes de error en caso de infringir alguna de estas reglas.

[23] *Pencil Project*: herramienta gratuita para la creación de bocetos a partir de un conjunto de figuras identificadas. Proporciona herramientas de prototipado de forma gratuita y libre, para realizar las maquetas en plataformas.

La vista de la página para iniciar sesión se muestra en la figura 2.5:

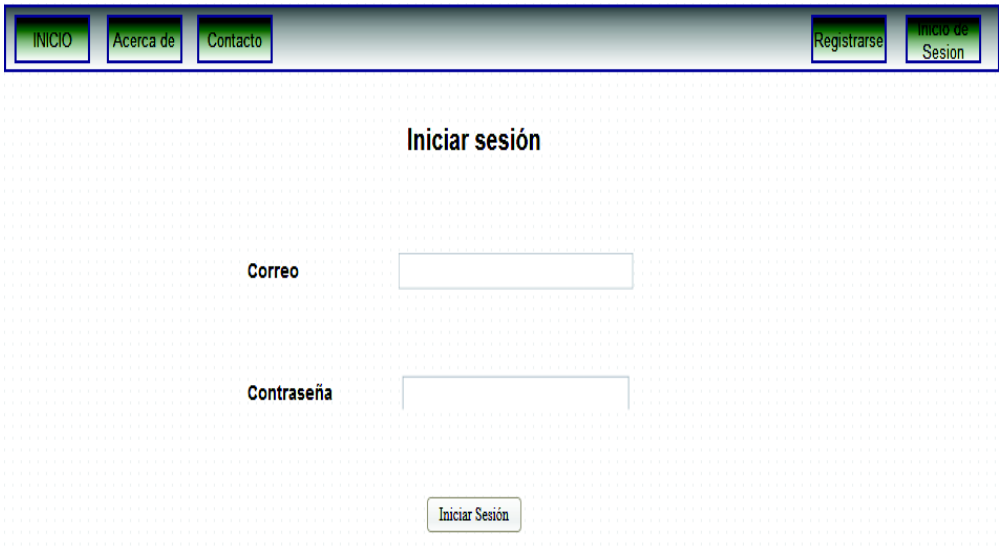


Figura 2.5 Vista inicio de sesión

- **Proceso de inicio de sesión**

Para el proceso de autenticación o inicio de sesión, se ha diseñado el diagrama de actividades presentado en la figura 2.6, que explica de manera detallada los pasos que se ejecutan internamente en la aplicación.

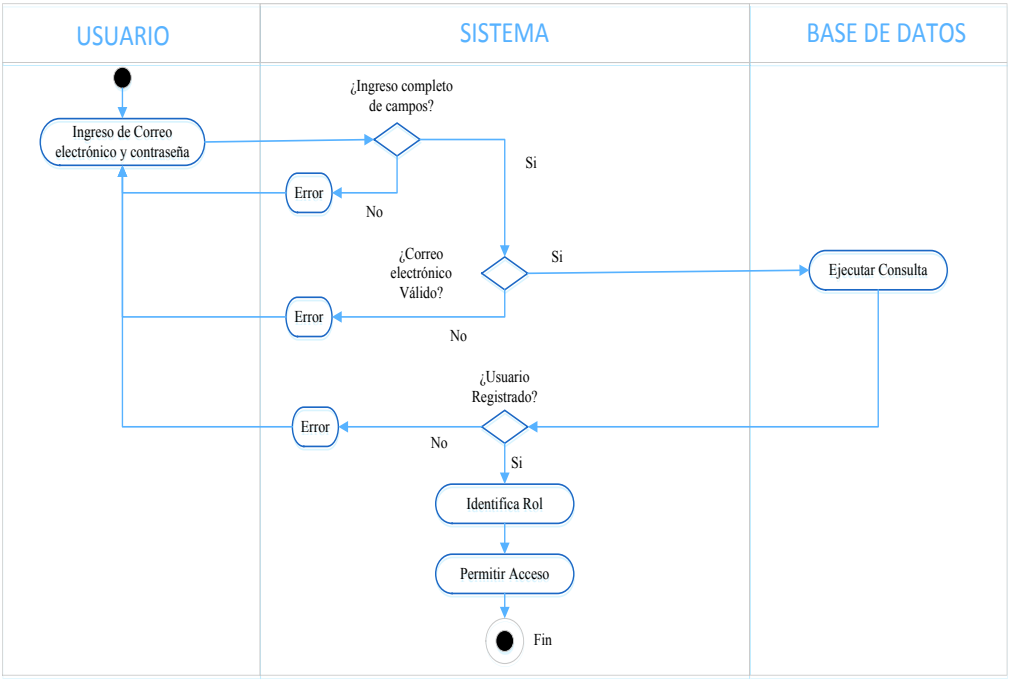


Figura 2.6 Proceso de inicio de sesión

2.2.1.4.2 Módulo de Administración de Usuarios

A continuación se presenta la vista para realizar la gestión de los usuarios, esta muestra una lista de todos los usuarios registrados en la aplicación, en la cual se puede observar los nombres, correo, teléfono, dirección, y la foto.

La vista de la gestión de usuarios se muestra en la figura 2.7

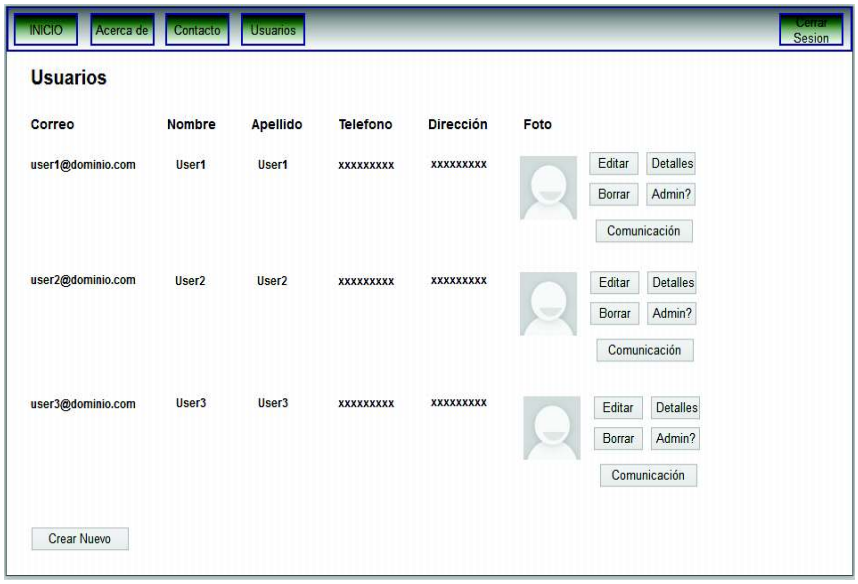


Figura 2.7 Vista del administrador para la gestión de usuarios

- **Vista para la creación de usuarios**

Una vez registrada la persona, se le asigna el rol de usuario. La vista de la página para crear usuarios se muestra en la figura 2.8:

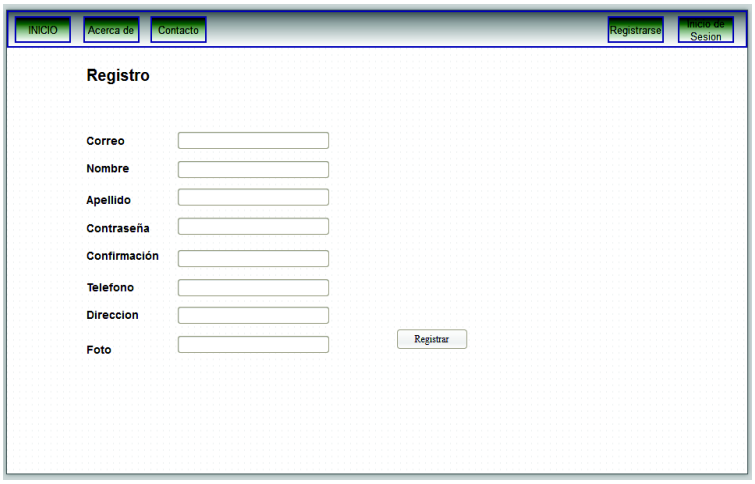


Figura 2.8 Vista para la creación de usuarios

- **Proceso para la creación de usuarios**

Para el proceso de registro o creación de usuarios, se ha diseñado el diagrama de actividades presentado en la figura 2.9, que explica de manera detallada los pasos que se ejecutan internamente.

Como se explicó en la anterior sección la clave foránea[24] de la tabla *Users* es el campo del correo electrónico, debido a esto, existirá una restricción en el proceso de registro que no permite crear dos usuarios con la misma dirección de correo electrónico.

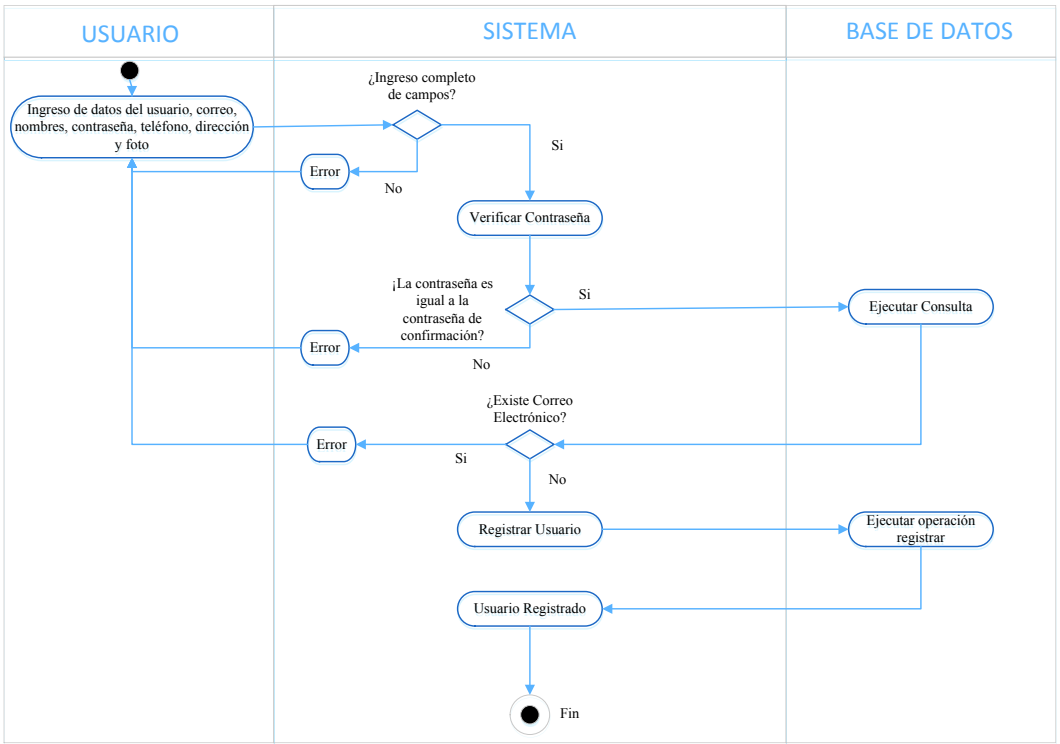


Figura 2.9 Proceso para la creación de usuario

- **Vista para editar la configuración de un usuario**

La primera vista permite editar la información personal del usuario, es decir, sus nombres, teléfono, dirección, foto y la segunda vista permite editar la contraseña. En el caso del cambio de la contraseña, de igual manera, se debe comparar que el

[24] Clave foránea: Identifica una columna o grupo de columnas en una tabla, que se refiere a una columna o grupo de columnas en otra tabla.

campo de la contraseña nueva y el campo para la verificación de la contraseña sean iguales.

Se notifica al usuario con mensajes de error en caso de infringir alguna regla.

La vista de la página para editar el perfil se muestra en figura 2.10 y la vista para editar la contraseña se muestra en la figura 2.11, de esta manera se puede editar la información personal:

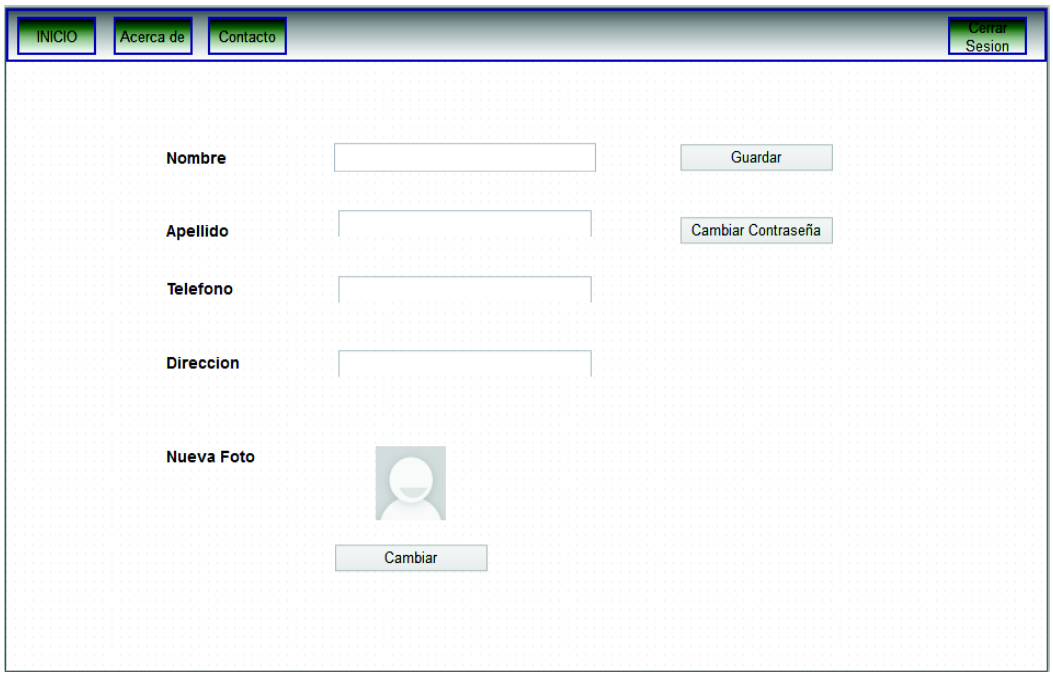


Figura 2.10 Vista para editar la configuración personal

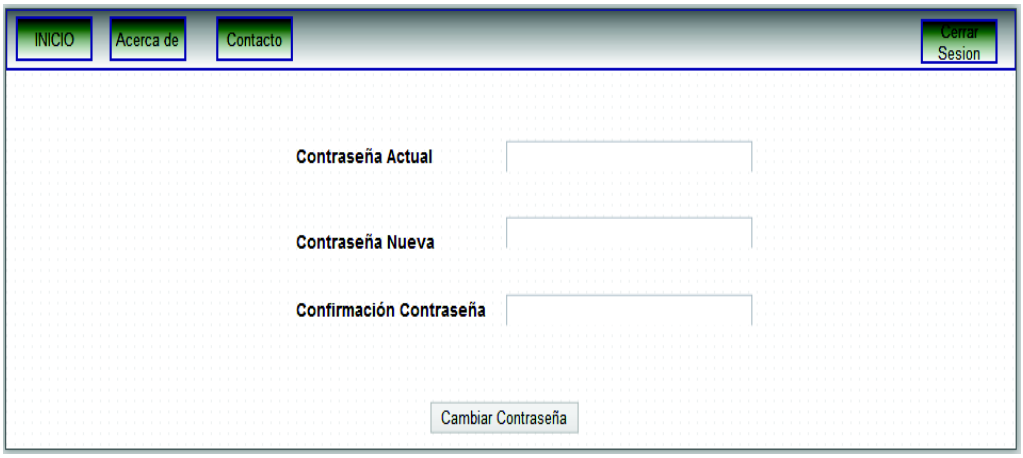


Figura 2.11 Vista para editar la contraseña

- **Proceso para editar un usuario**

En el proceso para editar un usuario en la aplicación, no se permite editar el campo del correo electrónico, Se ha diseñado el diagrama de actividades presentado en la figura 2.12, que explica de manera detallada los pasos que se ejecutan internamente.

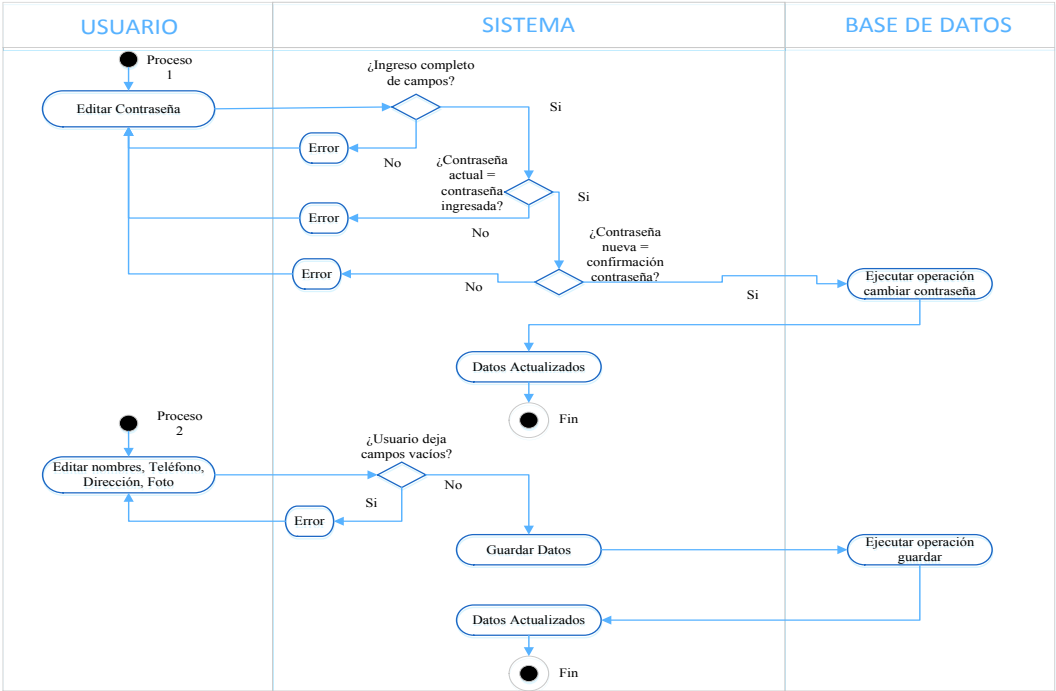


Figura 2.12 Proceso para editar un usuario

- **Vista para eliminar un usuario**

La vista para eliminar usuarios, se muestra en la figura 2.13. Esta vista funciona como una confirmación para eliminar usuarios de la aplicación.

Figura 2.13 Vista para eliminar un usuario

- **Proceso para eliminar un usuario**

Para el proceso de eliminación de un usuario, se ha diseñado el diagrama de actividades presentado en la figura 2.14, que explica de manera detallada los pasos que se ejecutan internamente en la aplicación.

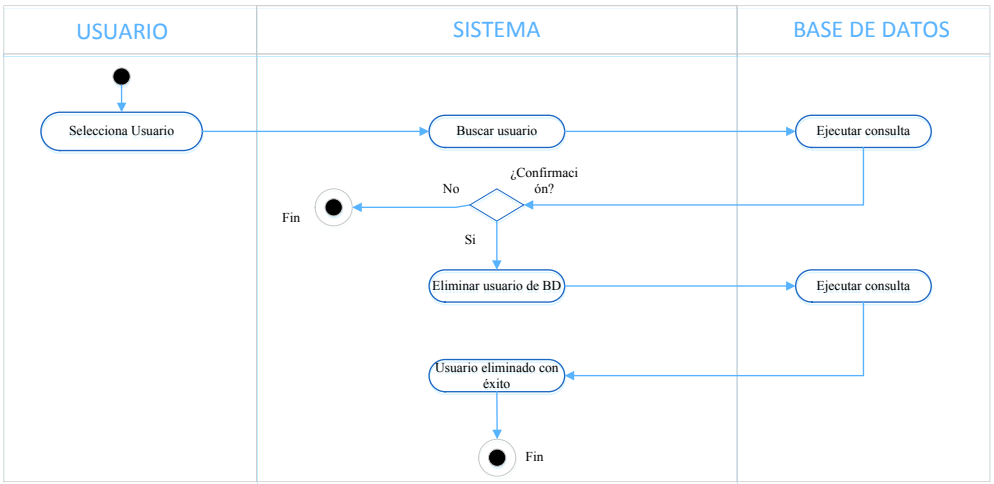


Figura 2.14 Proceso para eliminar un usuario

- **Proceso para cambiar de rol**

Para el proceso agregar o remover el rol de administrador, se ha diseñado el diagrama de actividades presentado en la figura 2.15, que explica de manera detallada los pasos que se ejecutan internamente en la aplicación:

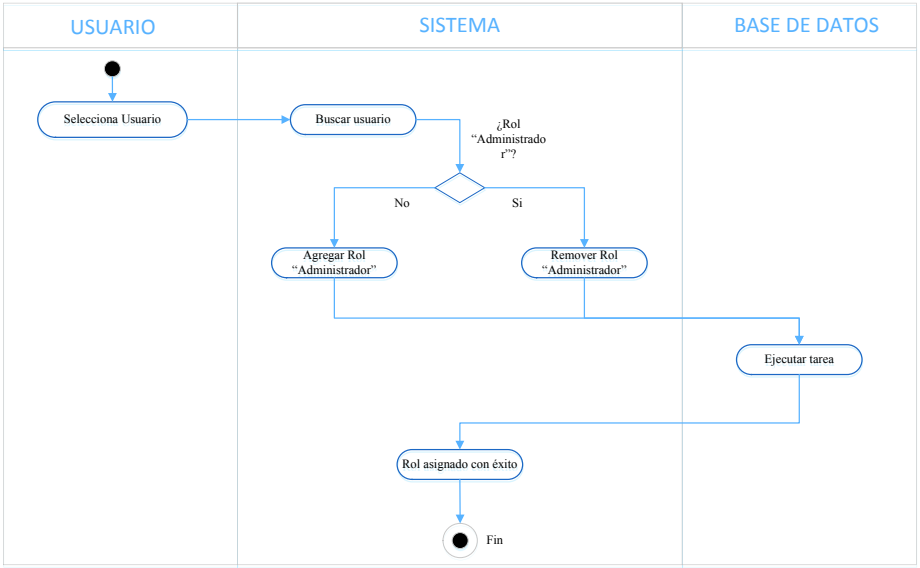


Figura 2.15 Proceso para cambiar el rol

Si un persona tiene el rol de usuario, al ejecutar este proceso se le asigna los dos tipos de roles, en cuyo caso, la diferencia entre cada rol es el nivel de permisos para manipular los recursos de la aplicación.

Si por el contrario la persona tiene el rol de administrador y se ejecuta este proceso, se le está quitando el nivel de permisos que tiene en la aplicación.

2.2.1.4.3 Módulo de Administración de Grupos

Los grupos permiten administrar de mejor manera una lista de usuarios, para de esta forma tener mejor ordenamiento de los usuarios registrados.

Por ejemplo se podrían tener grupos como:

- Administración
- Contabilidad
- Técnica
- Mantenimiento
- Monitoreo
- Estudiantil

Con la vista mostrada en la figura 2.16, el administrador puede realizar el proceso de CRUD de los grupos de la aplicación, y de igual manera sobre los miembros que conforman cada grupo.

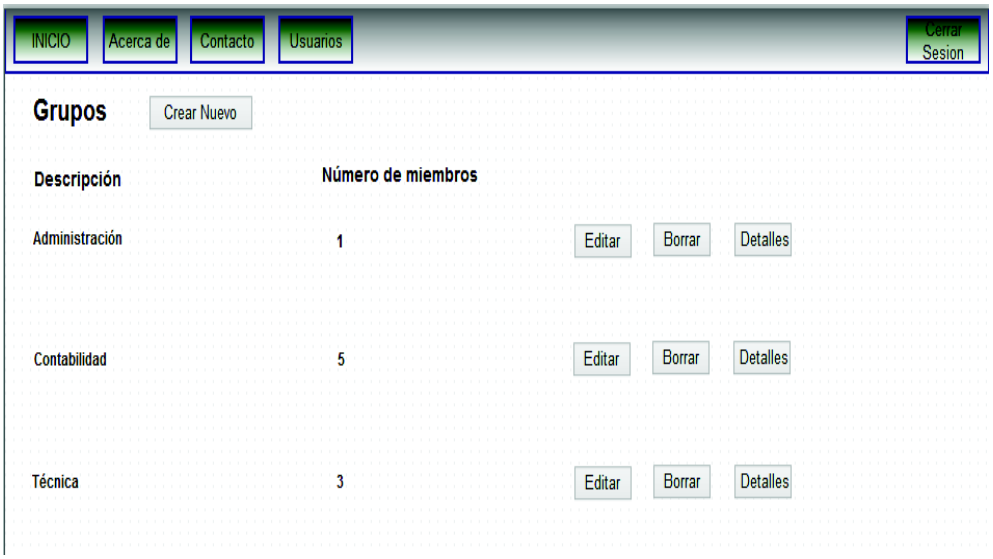


Figura 2.16 Vista del administrador para la gestión de grupos

- **Vista para la creación de usuarios**

La vista para crear un grupo se muestra en la figura 2.17, como se puede observar únicamente tiene el campo descripción.

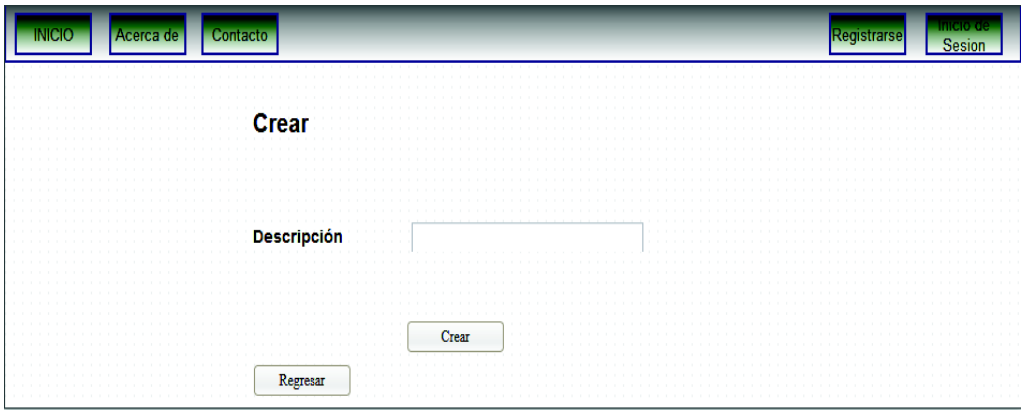


Figura 2.17 Vista para la creación de usuarios

- **Proceso para crear grupos**

Para el proceso de creación de grupos, que permiten un mejor ordenamiento de los usuarios registrados en la aplicación, se ha diseñado el diagrama de actividades presentado en la figura 2.18, que explica de manera detallada los pasos que se ejecutan internamente.

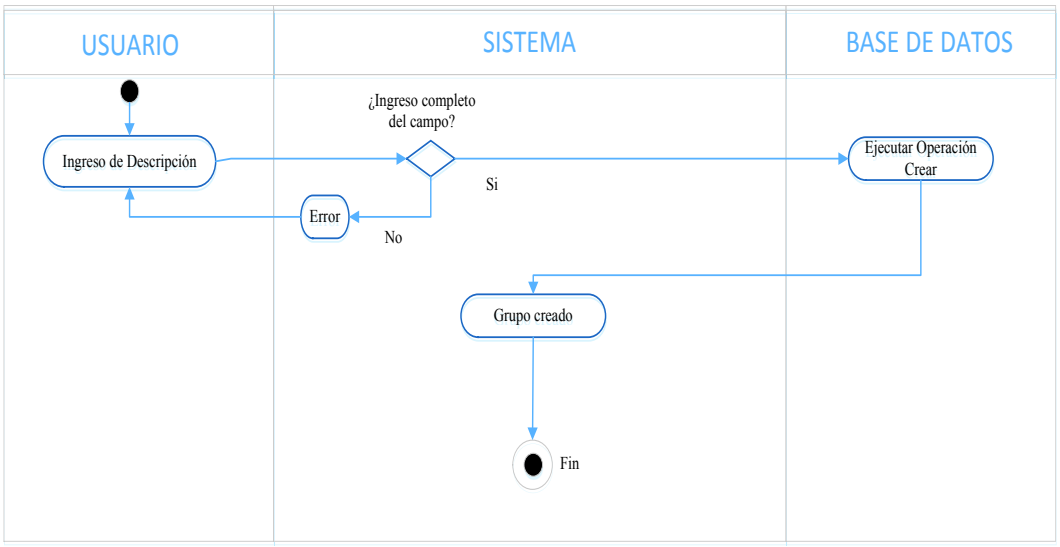


Figura 2.18 Proceso para la creación de un grupo

- **Vista para eliminar un grupo**

La vista para eliminar un grupo se muestra en la figura 2.19.

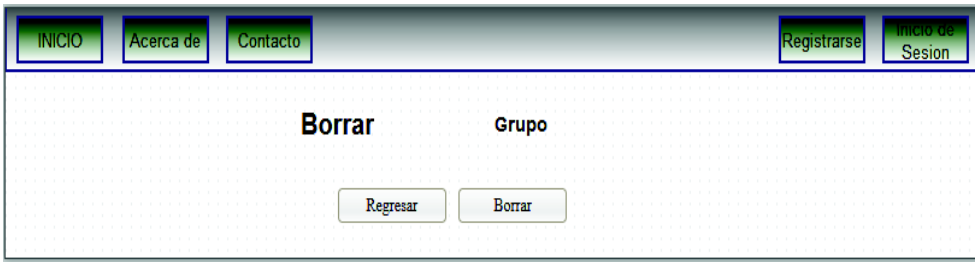


Figura 2.19 Vista para eliminar un grupo

- **Proceso para eliminar un grupo**

Para el proceso de eliminación de grupos, se ha diseñado el diagrama de actividades presentado en la figura 2.20, que explica de manera detallada los pasos que se ejecutan internamente.

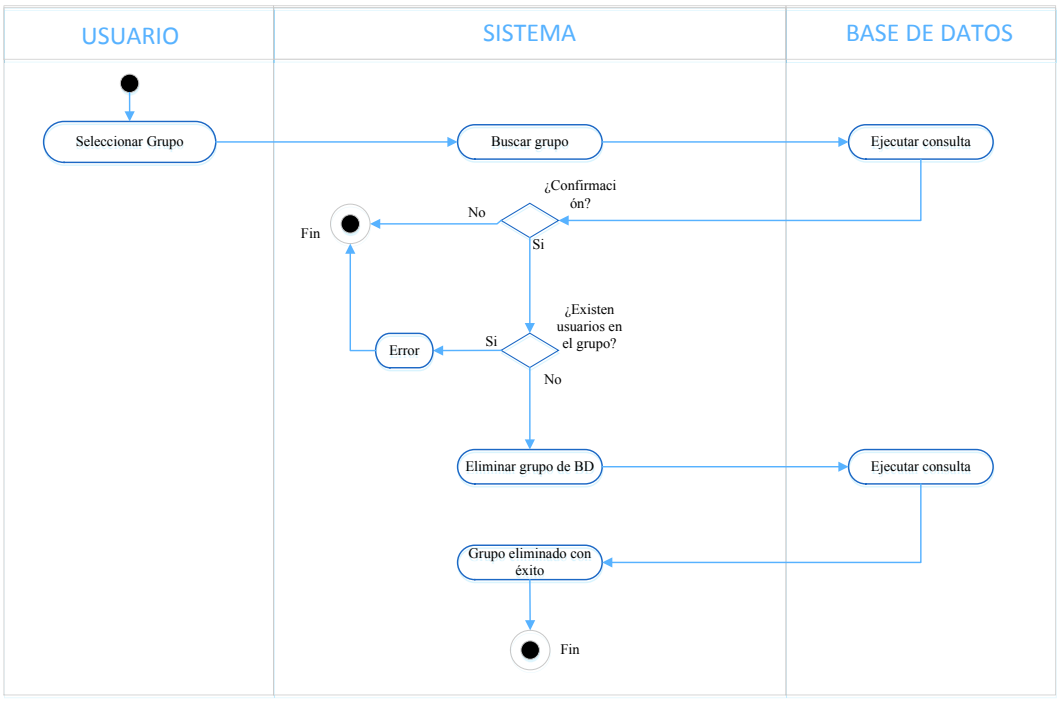


Figura 2.20 Proceso para eliminar un grupo

- **Vista para editar un grupo**

La vista para editar la descripción de un grupo se muestra en la figura 2.21. No pueden contener campo vacío.

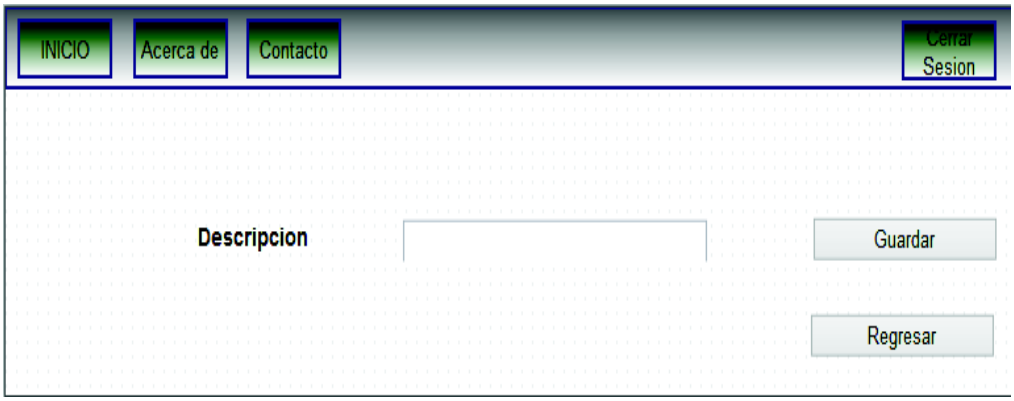


Figura 2.21 Vista para editar un grupo

- **Proceso para editar un grupo**

En el proceso para editar la descripción de un grupo, se ha diseñado el diagrama de actividades presentado en la figura 2.22, que explica de manera detallada los pasos que se ejecutan internamente.

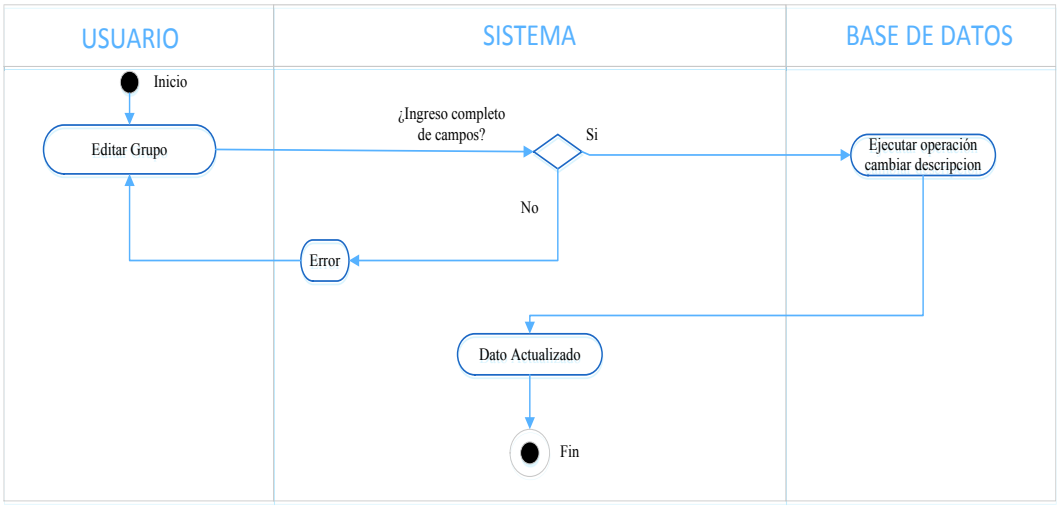


Figura 2.22 Proceso para editar un grupo

- **Vista para añadir miembros al grupo**

La vista para añadir miembros a un grupo permite agregar a cada grupo creado un miembro, de tal forma que los usuarios, pueden pertenecer a más de un grupo si así se desea, el diagrama de procesos permite observar cómo se realizan internamente las tareas para poder agregar los miembros se muestra en la figura 2.23.

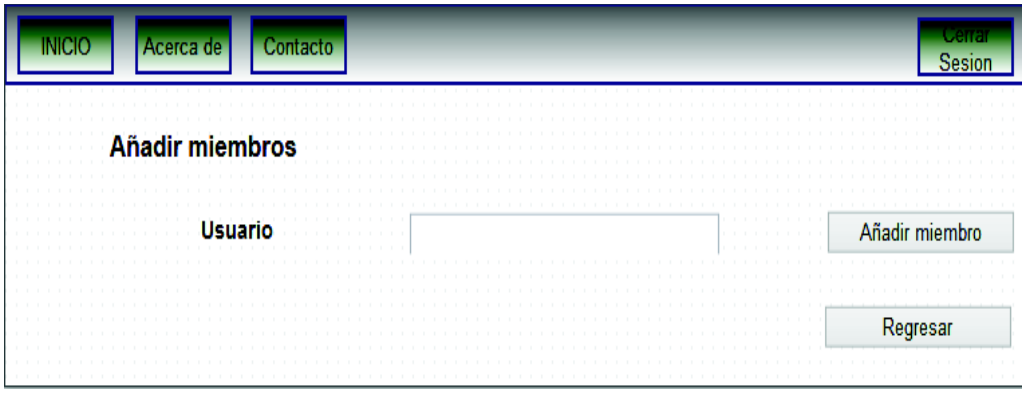


Figura 2.23 Vista para añadir miembros

- **Proceso para añadir miembros al grupo**

Para el proceso añadir miembros al grupo, se ha diseñado el diagrama de actividades presentado en la figura 2.24, que explica de manera detallada los pasos que se ejecutan internamente en la aplicación:

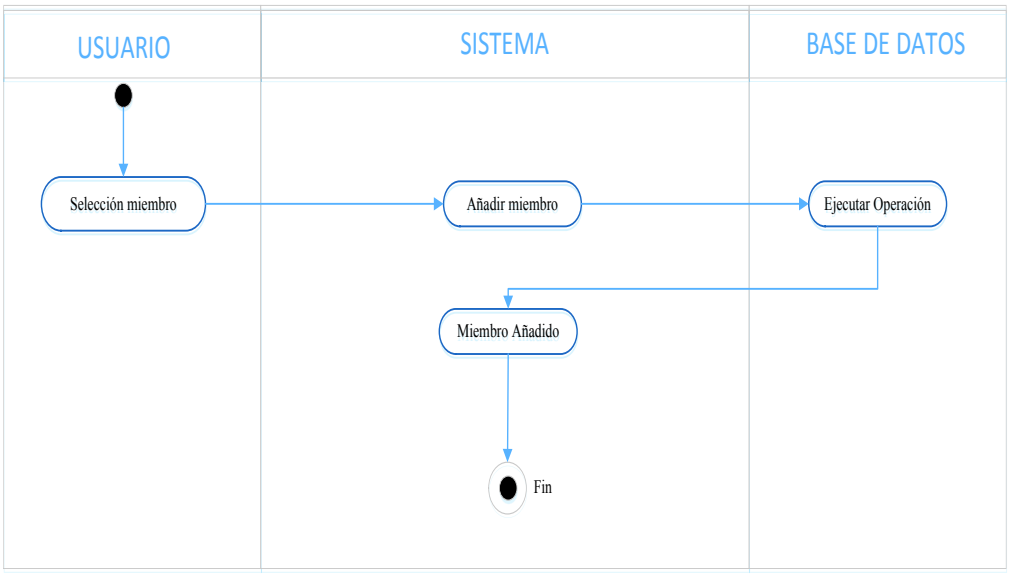


Figura 2.24 Proceso para añadir miembros

- **Vista para eliminar miembros del grupo**

La vista para eliminar miembros a un grupo se muestra en la figura 2.25, donde se puede observar que únicamente se necesita confirmación.

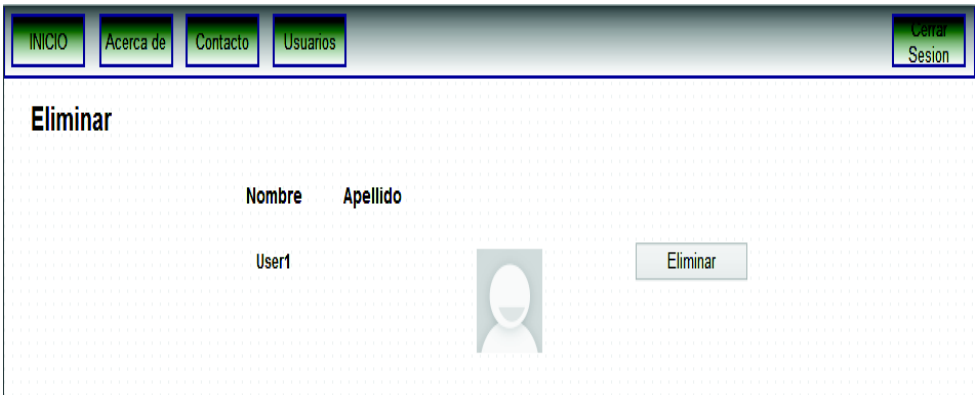


Figura 2.25 Vista para eliminar miembros

- **Proceso para eliminar miembros del grupo**

Para el proceso eliminar miembros del grupo, se ha diseñado el diagrama de actividades presentado en la figura 2.26, que explica de manera detallada los pasos que se ejecutan internamente en la aplicación.

El proceso para eliminar un miembro, se representa como una confirmación en la página, es decir, es un evento dinámico, luego se realiza la consulta a la base de datos para eliminar el registro respectivo.

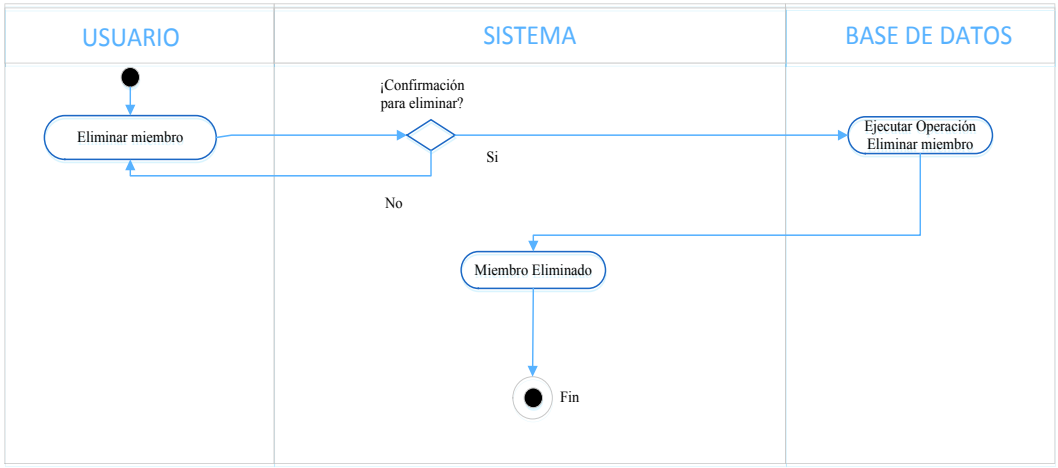


Figura 2.26 Proceso para eliminar miembros

2.2.1.4.4 Módulo de Comunicación

En el módulo de comunicación un usuario podrá realizar la comunicación en tiempo real con otro usuario registrado, en el que podrá realizar una videollamada, enviar mensajes y transferir archivos permitidos por el aplicativo web.

- **Página para la comunicación**

La vista en la que se puede realizar la comunicación en tiempo real, es un tipo de vista especial, ya que será diferente a las otras. Se tiene una página web desarrollada para que se ajuste al tamaño de pantalla en el que se visualice mediante el uso de Bootstrap, únicamente se agregan las líneas de código donde se llama al *framework.*

Las diferencias radican en que será desarrollada con HTML5, JavaScript, CSS3 y se usarán las API de WebRTC. La página de comunicación, se muestra en la figura 2.27.

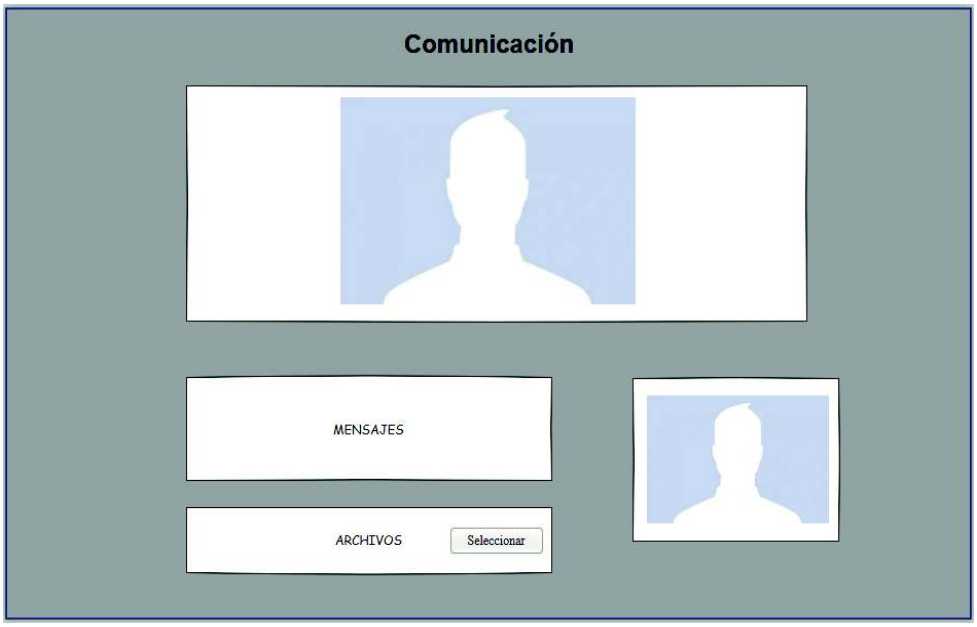


Figura 2.27 Página para la comunicación

- **Proceso de comunicación**

A continuación se presenta en la figura 2.28 un esquema con el funcionamiento de la comunicación, suponiendo que el usuario ya ha iniciado sesión, y además se ha ingresado exitosamente a la aplicación, se tiene un perfil creado y la comunicación se la realiza con otro usuario en línea, de tal forma que se pueda enviar el contexto de datos, para que ambos usuarios estén en la misma sala de comunicación.

Si por el contrario, se quiere estar en otra sala de comunicación se debe utilizar un contexto de datos diferente:

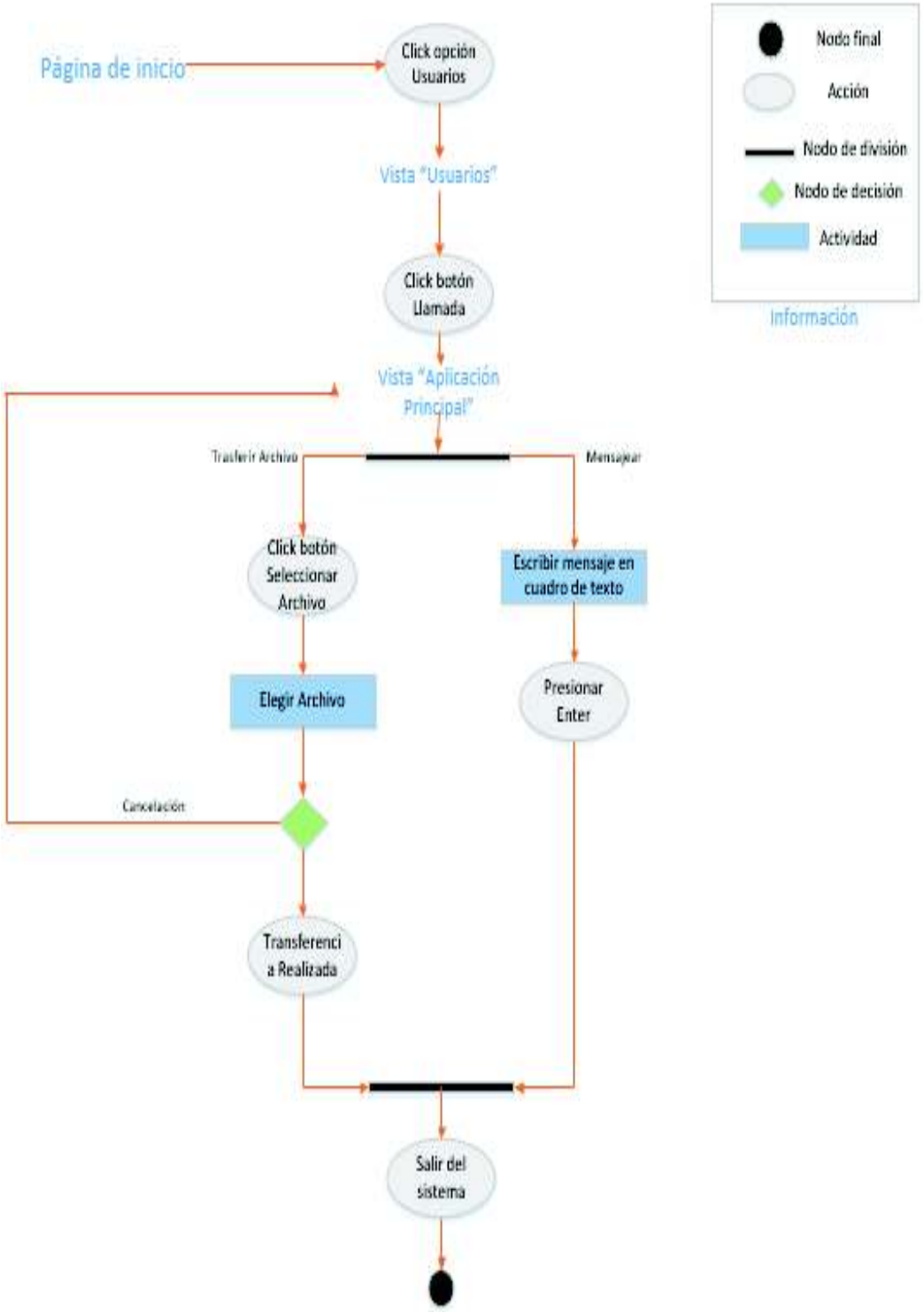


Figura 2.28 Proceso de comunicación

El proceso descrito sirve tanto para un usuario como para un administrador.

- **Diagrama de secuencia del proceso de comunicación**

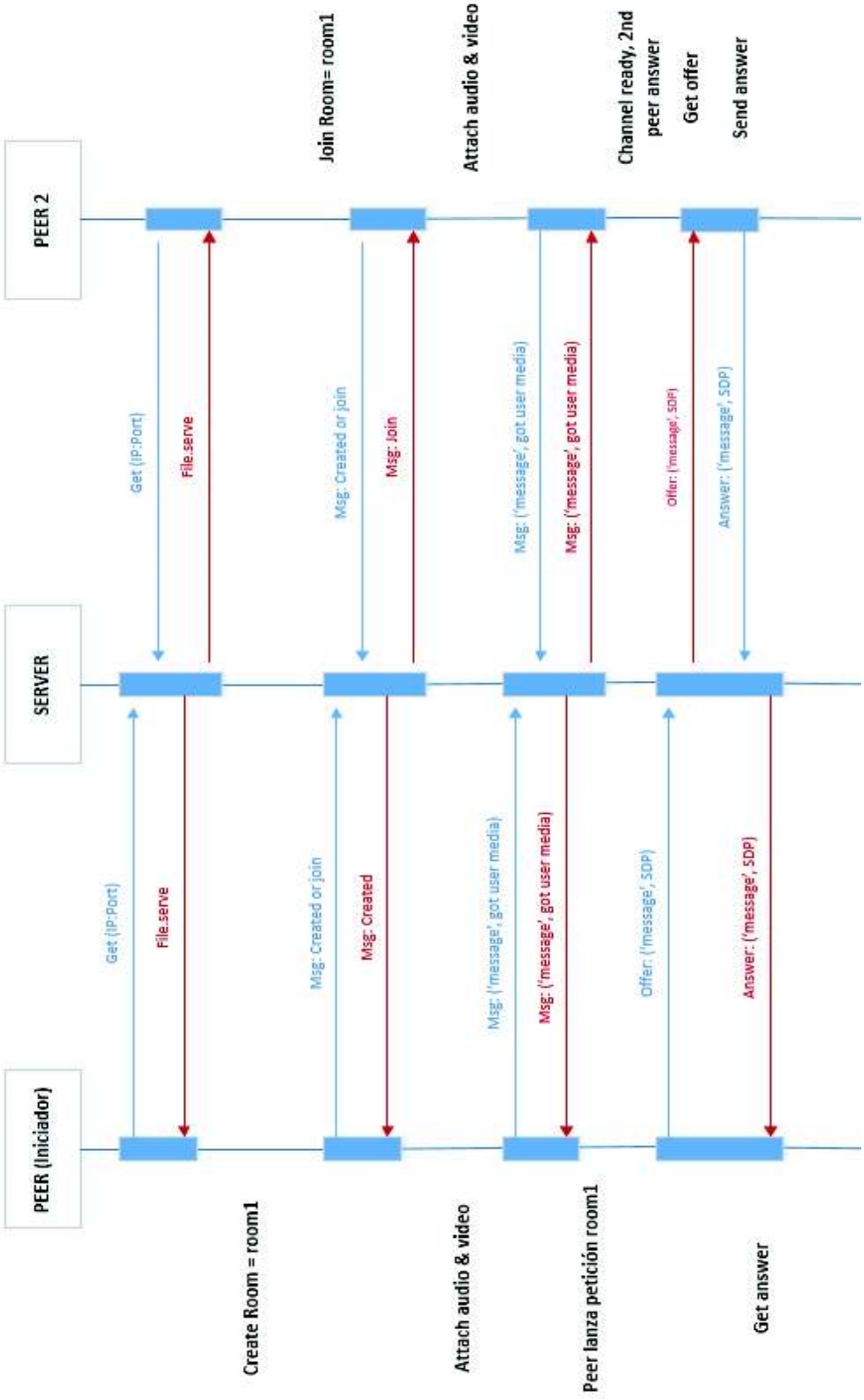


Figura 2.29 Diagrama de secuencia

3 CAPÍTULO 3: IMPLEMENTACIÓN Y EVALUACIÓN

En este capítulo se presenta la implementación y las pruebas que verifican el correcto funcionamiento de cada uno de los módulos que conforman la aplicación web.

3.1 IMPLEMENTACIÓN DE LA APLICACIÓN

3.1.1 TECNOLOGÍAS DEFINIDAS

Para desarrollar aplicaciones web – MVC en Visual Studio, se realizan los siguientes pasos:

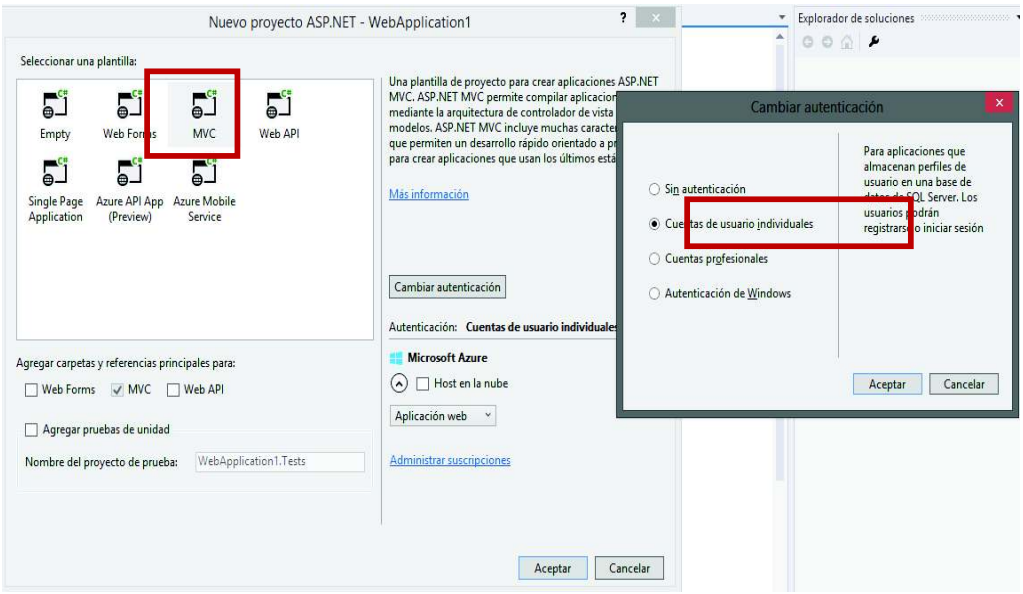


Figura 3.1 MVC en aplicaciones web

- Crear el proyecto web.
- Seleccionar la plantilla MVC en base a la cual se crea el proyecto.
- Elegir la opción de cuentas de usuario individuales que permite almacenar los perfiles de usuario en la base de datos SQL y también proporciona seguridad a través del uso del espacio de nombres *System.Web.Security*.

En la figura 3.2 se puede observar las carpetas que por defecto crea el sistema Visual Studio, usando la plantilla MVC, donde además se puede observar las clases que trae incorporado.

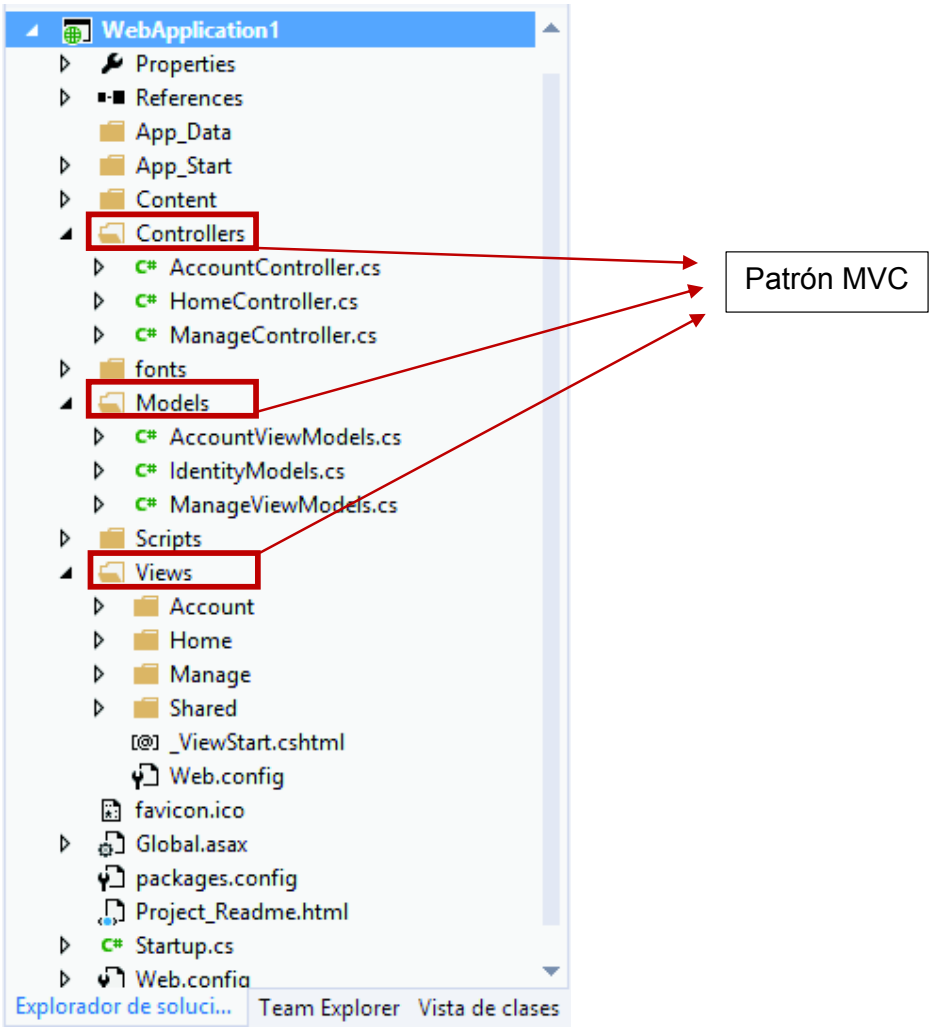


Figura 3.2 Patrón MVC

Luego de creado el proyecto, lo primero que se debe realizar, es cambiar la cadena de conexión a la base de datos en el archivo *Web.config*. Cabe mencionar que no hace falta crear la base de datos en SQL, ya que *Entity Framework*, examina si hay una base de datos con el nombre proporcionado, si no existe, la crea.

3.1.1.1 Métodos de Acción [20]

Los métodos de acción tienen normalmente una asignación unívoca con las interacciones del usuario. Por ejemplo las acciones pueden ser:

- Especificar una dirección URL en el Navegador.
- Hacer *click* en un vínculo.
- Enviar un formulario.

Cada una de estas interacciones del usuario, produce el envío de una solicitud al servidor. Cuando un usuario introduce una dirección URL en el explorador, la aplicación MVC usa reglas de enrutamiento que están definidas en el archivo *Global.asax*[25] para analizar la dirección URL y determinar la ruta de acceso del controlador.

Luego, el controlador determina el método de acción adecuado para administrar la solicitud. De forma predeterminada, la dirección URL de una solicitud se trata como una subruta de acceso que incluye el nombre del controlador seguido por el nombre de la acción.

3.1.2 BASE DE DATOS

El sistema gestor de base de datos usado en este proyecto es SQL Server 2012, que permite almacenar la información referente a los usuarios. Se utilizó este sistema ya que, permite una conexión de manera más directa y óptima con Visual Studio[26], [21].

De esta manera se logra una reducción considerable del retardo existente para realizar una consulta.

Como se mencionó en la sección 2.3.1.8, las tablas se crean a medida que se desarrollan las clases en C#, únicamente se debe ingresar la cadena de conexión a la base de datos desde el lado de la aplicación.

De forma predeterminada, los proyectos *Aplicaciones Web* usan una base de datos pequeña, que almacena la información dentro de la misma solución. Se ha cambiado la cadena de conexión hacia la base de datos ya que se utiliza una de mejores características, la conexión a la base de datos se muestra en la figura 3.3:

```
<connectionStrings>
  <add name="DefaultConnection"
       connectionString="Data Source=.;Initial Catalog=primerModulo;Integrated Security=True"
       providerName="System.Data.SqlClient" />
</connectionStrings>
```

Figura 3.3 Cadena de conexión a la base de datos

[25] *Global.asax*: Es un archivo opcional que contiene código para responder a eventos en el nivel de aplicación provocados por ASP .NET.
[26] Visual Studio: Conjunto completo de herramientas de desarrollo para la generación de aplicaciones web ASP.NET, servicios web XML, app de escritorio y app para móviles.

3.1.3 INSTALACIÓN DE XSOCKETS

Para instalar XSockets de manera sencilla al proyecto, se ingresa el comando mostrado en la figura 3.4, en el *Console Package Manager*[27]. [22]

```
PM> Install-Package XSockets -Version 5.0.2.2
```

Figura 3.4 Instalación de xsockets

Una vez instalado el paquete XSockets en el proyecto, se añaden las respectivas referencias que permiten el desarrollo del módulo de comunicación de la aplicación propuesta.

Las referencias que se añaden, se muestran en la figura 3.5.

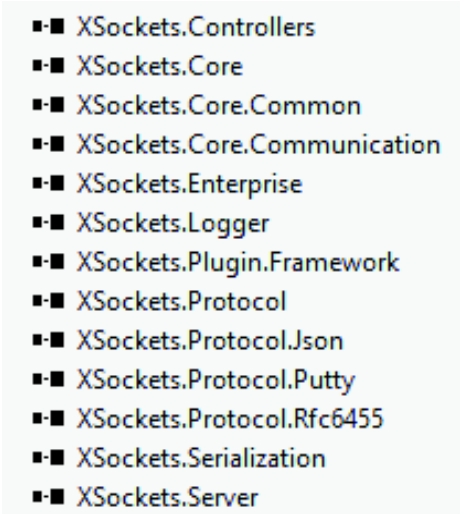


Figura 3.5 Referencias de xsockets

3.1.4 IMPLEMENTACIÓN DE LOS MÓDULOS

En la implementación de los módulos de la aplicación, se ha utilizado Anotaciones de Datos[28], que ha permitido personalizar las clases y asegurar el ingreso de datos válido. Las funciones de las anotaciones de datos definidas para este proyecto se adjuntan en el anexo B.

---

[27] *Console package manager*: Consola basada en PowerShell, que permite consultar, instalar, actualizar y eliminar paquetes de los proyectos de Visual Studio.
[28] *Data Annotations*: Especifican reglas de validación, establecen relaciones y especifica como se muestra los datos. Usado para personalizar las clases.

3.1.4.1 Implementación del Módulo de Autenticación

En la implementación del módulo de autenticación, se utilizó las tablas creadas por el *framework* ASP .NET, que proporciona las seguridades definidas en la sección 2.2.1.3, para el acceso a la aplicación. El módulo de autenticación permite que un usuario ingrese, proporcionando su correo electrónico y su contraseña.

3.1.4.1.1 Modelo

Se debe definir las restricciones de la aplicación en base al uso de las Anotaciones de datos. A continuación se presenta en la tabla 3.1, los atributos más importantes agregados a la clase con sus respectivas *Data Annotations*.

Tabla 3.1 *Data annotations* utilizadas para el inicio de sesión

Atributo	Data Annotations
Email	*Display, Required, EmailAddress*
Password	*Display, Required, DataType.*
RememberMe	*Display*

La anotación principal utilizada es *Required* que verifica que se ingrese un valor, es decir, que el campo que ingresa el usuario no este vacío. El código implementado para este modelo se lo puede observar en la figura 3.6.

```
1 referencia
public class LoginViewModel
{
    [Required]
    [Display(Name = "Correo electrónico")]
    [EmailAddress]
    0 referencias
    public string Email { get; set; }

    [Required]
    [DataType(DataType.Password)]
    [Display(Name = "Contraseña")]
    0 referencias
    public string Password { get; set; }

    [Display(Name = "¿Recordar cuenta?")]
    2 referencias
    public bool RememberMe { get; set; }
}
```

Figura 3.6 Modelo para el inicio de sesión

3.1.4.1.2 Controlador

El controlador procesa las solicitudes entrantes, controla los datos proporcionados por los usuarios y ejecuta la lógica de la aplicación adecuada.

Este controlador gestiona el proceso de autenticación del usuario, por ello recibe dos métodos de acción, una para cuando el usuario llama a la vista de inicio de sesión y otra cuando se envían los datos para la verificación. A continuación se presenta los dos métodos de acción creados:

El método de acción *GET*, se produce una vez que una persona presiona el botón "Iniciar Sesión" en la página de inicio, el controlador ejecuta la tarea necesaria para mostrar la vista, la implementación de este método se lo puede observar en la figura 3.7:

```
// GET: /Account/Login
[AllowAnonymous]
0 referencias
public ActionResult Login(string returnUrl)
{
    ViewBag.ReturnUrl = returnUrl;
    return View();
}
```

Figura 3.7 Método get del controlador

El atributo de clase *AllowAnonymous* permite omitir ciertas acciones de seguridad ya sea de autorización o de autenticación, en este caso permite el acceso al método a cualquier persona, sin necesidad de que tenga asignado un rol.

El método *POST* del controlador valida que el intento de inicio de sesión sea exitoso. El proceso para permitir que un usuario inicie sesión es el siguiente:

- El controlador verifica que sea un modelo válido, caso contrario simplemente devuelve la misma vista al usuario y no ejecuta ninguna acción.
- Verifica que el estado de inicio de sesión sea exitoso. Si la verificación falla, se muestra un mensaje de error.

- Si la verificación es exitosa, el usuario es redirigido a la vista de la página de inicio.

El código implementado para el controlador se lo puede observar en el anexo C.

3.1.4.1.3 Vistas

La vista “Inicio de sesión” le permite al usuario ingresar sus datos (correo electrónico y contraseña). La vista de inicio de sesión se la puede observar en la figura 3.8.

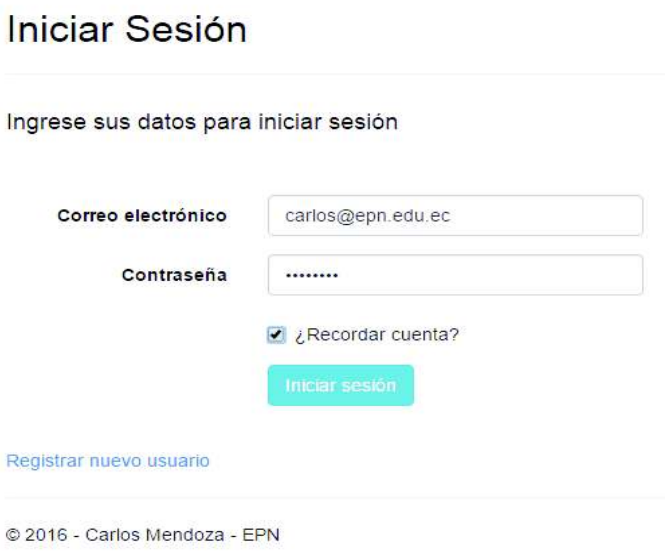


Figura 3.8 Vista inicio de sesión

3.1.4.2 Implementación del Módulo de Administración de Usuarios

3.1.4.2.1 Modelo

En la implementación de este módulo se ha definido más de un modelo para realizar las acciones de CRUD, y las vistas son diferentes para cada método de acción. A continuación se presenta la implementación de la clase *User*:

- Clase *User*

En la clase *User* se ha definido los atributos generales que son almacenados en la base de datos, con sus respectivos *Data Annotations*.

Estos se los puede observar en la tabla 3.2:

Tabla 3.2 *Data Annotation* para la clase user

Atributo	Data Annotations
UserId	*Key*
UserName	*Display, Required, StringLength, DataType, Index.*
FirstName	*Display, Required, StringLength.*
LastName	*Display, Required, StringLength.*
Full Name	*Display*
Phone	*Required, StringLength.*
Address	*Required, StringLength.*
Photo	*Required, DataType*

3.1.4.2.2 Controlador

El controlador debe procesar cada uno de los métodos de acción que realiza el usuario, a continuación se presentan los métodos implementados para las acciones realizadas por el usuario:

- ***Create***

  El método *Create* permite al administrador crear perfiles, asignado el rol de usuario. Para el método *GET*, el controlador recibe la acción realizada y muestra la vista respectiva. A continuación se presenta en la figura 3.9 la implementación del método para crear un nuevo perfil:

```
// GET: Users/Create
[Authorize(Roles = "Admin")]
0 referencias
public ActionResult Create()
{
    return View();
}
```

Figura 3.9 Implementación del método *create*

El atributo de clase *Authorize*[29] define que la persona autorizada para acceder al método, únicamente es aquella que tenga el rol de administrador.

El método *POST* del controlador permite crear el usuario y almacena el perfil en la base de datos. Este método se ejecuta una vez que el administrador presiona el botón crear.

El proceso para almacenar un perfil en la base de datos es el siguiente:

- El controlador interpreta la acción que realiza el administrador.
- El controlador verifica que sea un modelo válido.
- Caso contrario simplemente devuelve la misma vista al administrador y no ejecuta ninguna acción.
- El controlador asigna un rol.
- Este rol es la de usuario, si es un auto registro, ya que si el administrador crea el perfil de usuario, se puede agregar el rol una vez creado.
- El perfil del usuario es almacenado en la base de datos, si no se presenta ningún error.
- Los errores son mostrados dinámicamente, algunos de estos, pueden ser error por entradas nulas o por no colocar un correo electrónico.
- Se muestra la vista con la lista de los usuarios registrados en la aplicación.

El código implementado para el controlador se lo puede observar en el anexo C.

- ***Edit***

El método *Edit* permite al administrador editar o modificar información perteneciente a los usuarios registrados.

El controlador recibe la acción realizada y muestra la vista respectiva. A continuación se presenta en la figura 3.10 la implementación del método para editar la información de un usuario.

[29] *Authorize*: Define reglas de autorización para manipulación de datos, esta se puede definir a nivel de clase o a nivel de método.

```
// GET: Users/Edit/5
[Authorize(Roles = "Admin")]
0 referencias
public ActionResult Edit(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }

    var user = db.Users.Find(id);

    if (user == null)
    {
        return HttpNotFound();
    }


    var userView = new UserView
    {
        Addrees = user.Addrees,
        FirstName = user.FirstName,
        LastName = user.LastName,
        Phone = user.Phone,
        UserId = user.UserId,
        UserName = user.UserName
    };


    return View(userView);
}
```

Figura 3.10 Implementación del método *edit*

El atributo de clase *Authorize* define que la persona autorizada para acceder al método, únicamente es aquella que tenga el rol de administrador.

El método *GET* del controlador, para el caso de modificación de datos, busca al usuario por su identificador único, en caso de no encontrarlo, se muestra un mensaje de error. Una vez que se muestre la vista para modificar los datos, los campos contendrán la última información almacenada exitosamente.

El método *POST* del controlador permite modificar la información personal del usuario y la actualiza en la base de datos. El proceso realizado en el controlador es el siguiente:

- El controlador interpreta la acción que realiza el administrador.
- El controlador verifica que sea un modelo válido, caso contrario simplemente devuelve la misma vista al administrador y no ejecuta.

- Se crea una variable donde se carga la información a modificar del usuario.
- Se guardan los cambios realizados y se almacenan en la base de datos.

El código implementado para el controlador se lo puede observar en el anexo C.

- ***Details***

El método *Details* permite al administrador ver detalles de los usuarios registrados. Los datos observables son el correo electrónico, nombre, apellido, teléfono, dirección y la foto.

El controlador recibe la acción realizada y muestra la vista respectiva. A continuación se presenta en la figura 3.11 la implementación del método para ver detalles de los usuarios:

```
// GET: Users/Details/5
[Authorize(Roles = "Admin")]
0 referencias
public ActionResult Details(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    User user = db.Users.Find(id);
    if (user == null)
    {
        return HttpNotFound();
    }
    return View(user);
}
```

Figura 3.11 Implementación del método *details*

En caso de que no exista el usuario del que se desea ver los detalles, se muestra una página de error. El usuario es ubicado por su identificador único.

- ***Delete***

El método *Delete* permite al administrador eliminar la información perteneciente a los usuarios registrados.

El controlador recibe la acción realizada y muestra la vista respectiva, que sirve como una confirmación para eliminar los perfiles de los usuarios.

A continuación se presenta en la figura 3.12 la implementación del método para eliminar la información de un usuario:

```
// GET: Users/Delete/5
[Authorize(Roles = "Admin")]
0 referencias
public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    User user = db.Users.Find(id);
    if (user == null)
    {
        return HttpNotFound();
    }
    return View(user);
}
```

Figura 3.12 Implementación del método *delete*

El método *GET* del controlador para eliminar un perfil, busca al usuario por su identificador único, en caso de no encontrarlo, se notifica con un error. En el método *POST* del controlador se encuentra implementado el código necesario para eliminar el perfil de un usuario. El proceso realizado en el controlador es el siguiente:

- El controlador interpreta la acción que realiza el administrador.
- Se ubica al usuario por su identificador único y se lo almacena en una variable.
- El usuario es removido de la base de datos.
- Se guardan los cambios realizados en la base de datos.

El código implementado para el controlador se lo puede observar en el anexo C.

- ***Is Admin***

El método *Is Admin* permite al administrador, cambiar el rol de usuario a administrador o viceversa. El controlador recibe un método de acción que permite habilitar o deshabilitar esta opción.

A continuación se presenta en la figura 3.13 la implementación del método para el cambio de rol asignado a un perfil:

```
[Authorize(Roles = "Admin")]
0 referencias
public ActionResult OnOffAdmin(int id)
{
    var user = db.Users.Find(id);
    if (user != null)
    {
        var userContext = new ApplicationDbContext();
        var userManager = new UserManager<ApplicationUser>(new UserStore<ApplicationUser>(userContext));
        var userASP = userManager.FindByEmail(user.UserName);
        if (userASP != null)
        {
            if (userManager.IsInRole(userASP.Id, "Admin"))
            {
                userManager.RemoveFromRole(userASP.Id, "Admin");
            }
            else
            {
                userManager.AddToRole(userASP.Id, "Admin");
            }
        }
    }
    return RedirectToAction("Index");
}
```

Figura 3.13 Implementación del método *is admin*

El proceso realizado en el controlador es el siguiente:

- El controlador interpreta la acción que realiza el administrador.
- Se ubica al usuario por su identificador único y se lo almacena en una variable. Esta variable permite diferenciar al usuario, las variables son almacenadas en memoria.
- Verificar si el usuario es o no administrador. En caso de ser administrador, se remueve de este rol.
- Caso contrario se agrega el rol de administrador.
- Se muestra la vista respectiva al administrador, para que realice las acciones CRUD sobre los usuarios.

3.1.4.2.3 Vistas

La vista principal para la gestión de usuarios se observa a continuación en la figura 3.14:

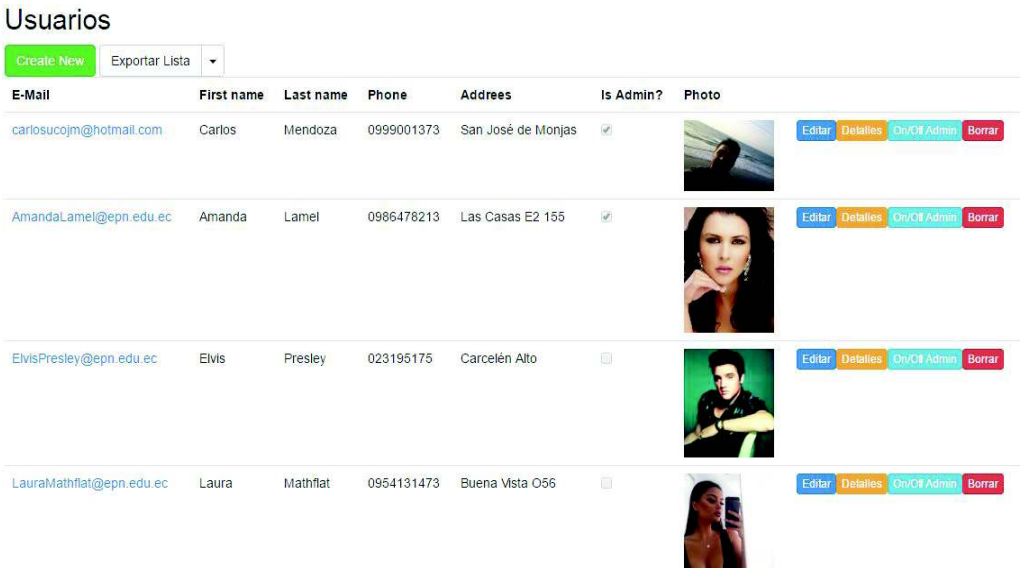


Figura 3.14 Vista para la administración de Usuarios

- **Vista *Create***

La vista *Create* se ha implementado para que el administrador pueda registrar usuarios. La misma se la puede observar en la figura 3.15.

Figura 3.15 Vista *create*

La diferencia existente entre la creación de una cuenta por el administrador, a la de auto registrarse, es que el administrador no ingresa contraseña en la cuenta creada,

la contraseña por defecto en este caso, es la dirección de correo electrónico del usuario.

- **Vista *Edit***

La vista *Edit* se ha creado para que el administrador, pueda modificar los datos de los usuarios registrados. La vista para editar se la puede observar en la figura 3.16.

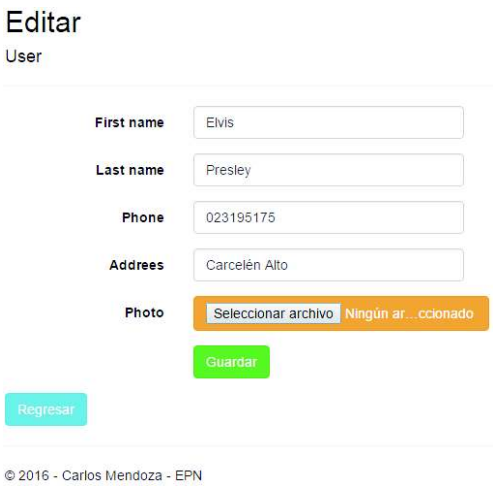


Figura 3.16 Vista *Edit*

- **Vista *Details***

La vista para ver los detalles de los usuarios se la puede observar en la figura 3.17.

Figura 3.17 Vista *details*

- **Vista *Delete***

La vista *Delete* se ha creado para que el administrador pueda eliminar el perfil de un usuario registrado. Esta vista se presenta como una confirmación para eliminar los datos almacenados en la base de datos. La vista para eliminar un perfil se la puede observar en la figura 3.18.

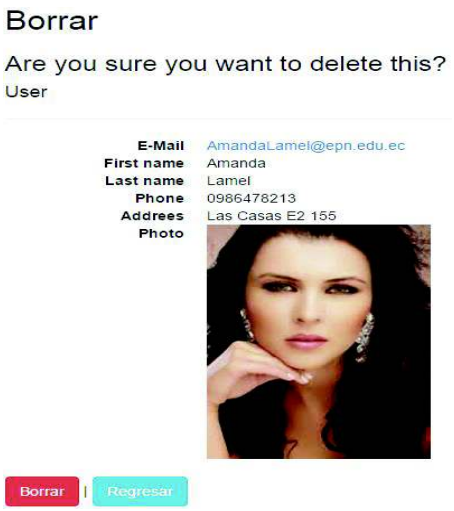


Figura 3.18 Vista *delete*

- **Vista *IsAdmin***

La vista donde se puede observar el rol asignado a un usuario, se presenta en la figura 3.19:

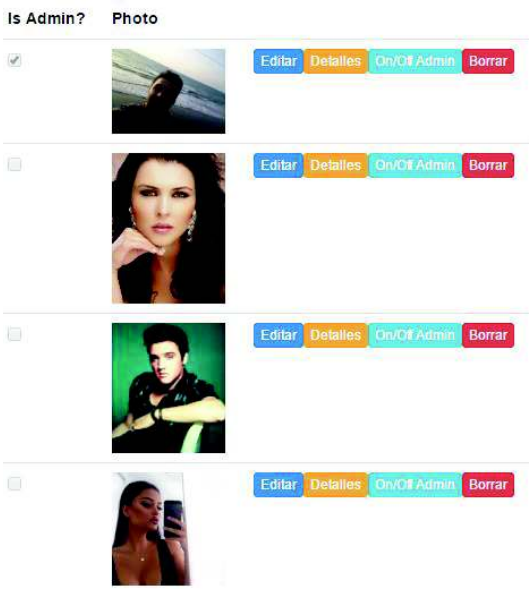


Figura 3.19 Ejemplo *isadmin*

En la figura anterior, se puede observar que únicamente el primer usuario, es administrador.

- **Vista *MySettings***

La vista *MySettings* permite al usuario o administrador editar sus propios datos, de esta manera se pueden hacer cambios sin tener la necesidad de que se encuentre el administrador de la aplicación para hacerlos. Esta vista se la puede observar en la figura 3.20 y la vista para modificar la contraseña se presenta en la figura 3.21.

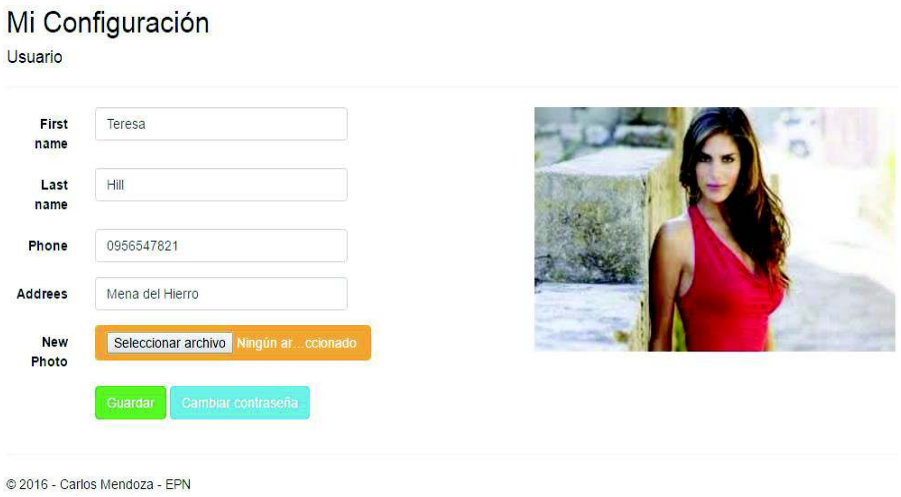


Figura 3.20 Vista *my settings*

Figura 3.21 Vista Cambio de Contraseña

3.1.4.3 Implementación del Módulo de Administración de Grupos

Este módulo permite realizar acciones CRUD sobre los grupos de usuarios.

3.1.4.3.1 Modelo

En la implementación de este módulo se ha definido más de un modelo para realizar las acciones de CRUD, y las vistas son diferentes para cada método de acción.

Las anotaciones de datos usadas para este módulo se las puede observar en la tabla 3.3:

Tabla 3.3 *Data Annotations* usadas para la implementación del módulo de administración de grupos

Atributo	***Data Annotations***
GroupId	*Key*
Decription	*Display, Required, StringLength.*

3.1.4.3.2 Controlador

El controlador debe procesar cada uno de los métodos de acción que realiza el administrador, a continuación se presentan los métodos implementados para las acciones realizadas:

- ***Create***

  El método *Create* permite al administrador crear grupos en la aplicación. Para el método *GET*, el controlador recibe la acción realizada y muestra la vista respectiva. A continuación se presenta en la figura 3.22 la implementación del método para crear un nuevo grupo:

```
// GET: Groups/Create
0 referencias
public ActionResult Create()
{
    return View();
}
```

Figura 3.22 Implementación del método *create*

El método *POST* del controlador permite crear un nuevo grupo y la descripción es almacenada en la base de datos.

El código implementado para el controlador se lo puede observar en el anexo C.

- ***Edit***

  El controlador recibe la acción realizada y muestra la vista respectiva. A continuación se presenta en la figura 3.23 la implementación del método para editar la información de un grupo:

```
// GET: Groups/Edit/5
0 referencias
public ActionResult Edit(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Group group = db.Groups.Find(id);
    if (group == null)
    {
        return HttpNotFound();
    }
    return View(group);
}
```

Figura 3.23 Implementación del método *edit*

El método *GET* del controlador, para el caso de modificación de datos de un grupo, busca al grupo por su identificador. Una vez que se muestre la vista para modificar los datos, los campos contendrán la última información almacenada exitosamente. El método *POST* del controlador permite almacenar los datos modificados de un grupo y los actualiza en la base de datos. El código implementado para el controlador se lo puede observar en el anexo C.

- ***Delete***

  El método *Delete* permite al administrador eliminar la información de un grupo, siempre y cuando no contenga miembros agregados, caso contrario, primero se debe eliminar sus miembros y luego el grupo.

El controlador recibe la acción realizada y muestra la vista respectiva, que sirve como una confirmación para eliminar un grupo.

A continuación se presenta en la figura 3.24 la implementación del método para eliminar un grupo:

```
// GET: Groups/Delete/5
0 referencias
public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Group group = db.Groups.Find(id);
    if (group == null)
    {
        return HttpNotFound();
    }
    return View(group);
}
```

Figura 3.24 Implementación del método *delete*

El método *GET* del controlador busca el grupo por su id, en caso de no encontrarlo, se notifica con un error.

En el método *POST* del controlador se encuentra implementado el código necesario para eliminar un grupo **y** se lo puede observar en el anexo C.

- ***AddMembers***

  El método *AddMember* permite al administrador agregar a usuarios a un determinado grupo. De esta forma se consigue crear grupos de trabajo que tengan relación unos con otros. Es decir, los usuarios que forman parte de un mismo grupo, serán capaces de observar que otros miembros forman parte de su mismo grupo, de esta manera aprovechar el uso del módulo de comunicación implementado en la siguiente sección, ya que si no forma parte de ningún grupo, el módulo de comunicación no se encontrará disponible.

  El controlador recibe la acción realizada y muestra la vista respectiva.

  A continuación se presenta en la figura 3.25 la implementación del método para agregar miembros a un grupo:

```
[HttpGet]
0 referencias
public ActionResult AddMember(int groupId)
{
    //Se lo ordena por nombres, si hay un empate se ordenan por apellidos
    ViewBag.UserId = new SelectList(db.Users
        .OrderBy(u => u.FirstName)
        .ThenBy(u => u.LastName), "UserId", "FullName");

    var view = new AddMemberView
    {
        GroupId = groupId,
    };

    return View(view);
}
```

Figura 3.25 Implementación del método *addmember*

- ***DeleteMembers***

El método *DeleteMember* permite al administrador eliminar usuarios de un determinado grupo. El controlador recibe la acción realizada y muestra la vista respectiva.

A continuación se presenta en la figura 3.26 la implementación del método para eliminar miembros de un grupo:

```
[HttpGet]
0 referencias
public ActionResult DeleteMember(int id)
{
    var member = db.GroupMembers.Find(id);
    if (member != null)
    {
        db.GroupMembers.Remove(member);
        db.SaveChanges();
    }

    return RedirectToAction(string.Format("Details/{0}", member.GroupId));
}
```

Figura 3.26 Implementación del método *deletemember*

3.1.4.3.3 Vistas

La siguiente vista permite al administrador realizar las acciones CRUD sobre los grupos de la aplicación: (figura 3.27)

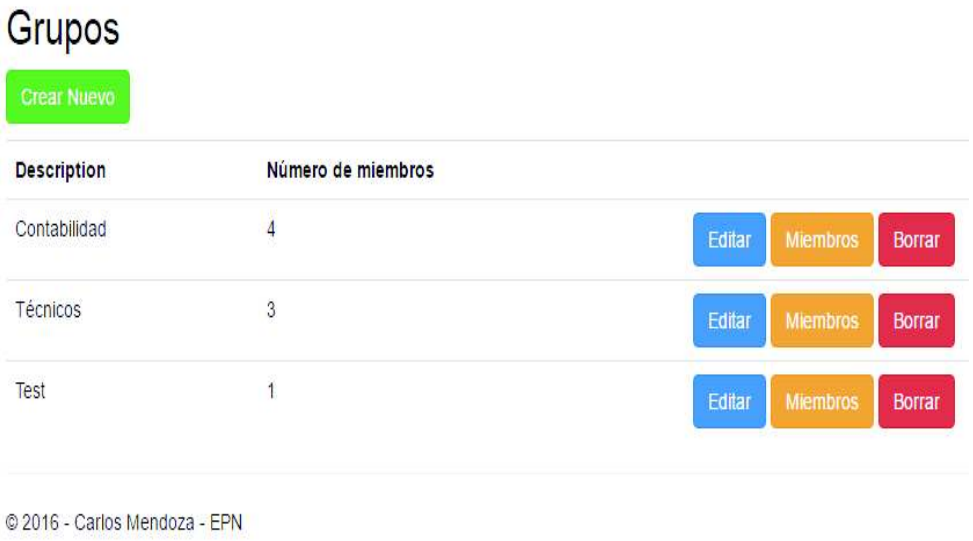


Figura 3.27 Vista para la gestión de los grupos

- **Vista *Create***

La vista *Create* permite crear un nuevo grupo, por ejemplo área de contabilidad, área de administración o área técnica; agregando para ello una descripción de este. Se lo puede observar en la figura 3.28.

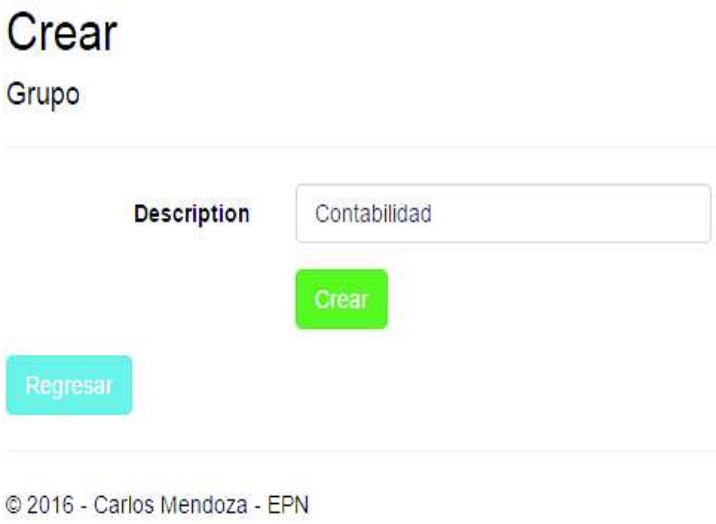


Figura 3.28 Vista *create* grupos

- **Vista *Edit***

La vista *Edit* permite editar la descripción de un grupo. La vista para editar se la puede observar en la figura 3.16.

Figura 3.29 Vista *edit* grupos

- **Vista *Delete***

La vista *Delete* permite eliminar un grupo de la aplicación, sin embargo, si el grupo contiene miembros, se los debe eliminar antes de eliminar el grupo.

La vista para eliminar un grupo se la puede observar en la figura 3.30.

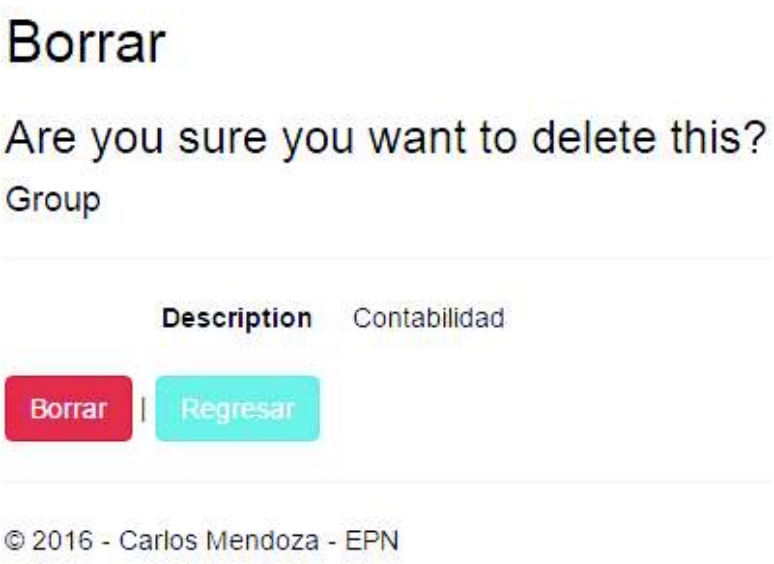


Figura 3.30 Vista *delete* grupos

- **Vista *AddMembers***

La vista *AddMembers* permite agregar miembros a un grupo, desplegando todos los miembros registrados en la aplicación.

La vista para añadir miembros se la puede observar en la figura 3.31.

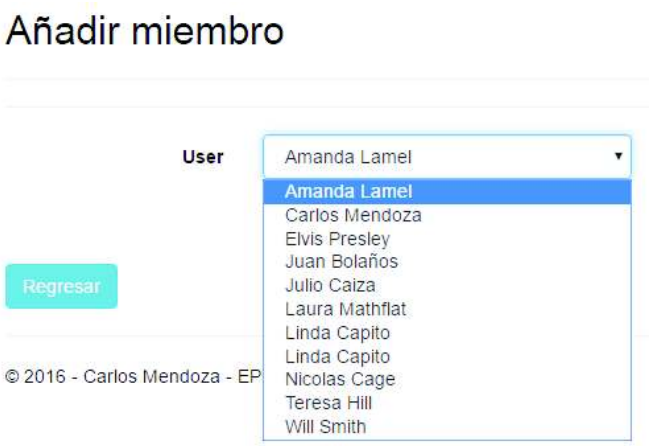


Figura 3.31 Vista *addmembers*

3.1.4.4 Implementación del Módulo de Comunicación

La implementación del módulo de comunicación se lo ha realizado de manera diferente a los anteriores módulos, ya que este ha sido desarrollado mediante HTML5, para la estructura de la página se han diseñado *scripts* para realizar algunas funcionalidades en JavaScript y se ha aprovechado varias librerías de XSockets que permiten la comunicación en WebRTC. El grupo de XSockets ha desarrollado varios ejemplos de comunicación en tiempo real con WebRTC, los cuales han sido publicados en la plataforma GitHub[30]. Para el desarrollo de este módulo se ha tomado componentes de algunos ejemplos de este repositorio, los cuales fueron modificados y adaptados. En especial se ha utilizado un ejemplo llamado “XSockets.WebRTC” [10] al cual se han podido despejar algunas dudas a lo largo del desarrollo, mediante consultas realizadas a los propietarios de esta plataforma. La modificación no infringe ninguna ley de propiedad intelectual en el desarrollo de este módulo.

A continuación se presentan los archivos principales que conforman el módulo de comunicación y las librerías que se importan, para desarrollar la funcionalidad y cumplir con los objetivos planteados.

[30] GitHub: Plataforma de desarrollo colaborativo, para alojar proyectos utilizando el sistema de control de versiones.

Archivos principales:

- *index*.html
- estiloTesis2.css

Librerías JavaScript utilizadas:

- XSockets.latest.js
- XSockets.WebRTC.latest.js

Los archivos anteriormente descritos, permiten la funcionalidad de la aplicación, a continuación se procede a la explicación de cada archivo definido, el código implementado se encuentra adjunto en el anexo C.

3.1.4.4.1 Index.html

Este archivo ha sido desarrollado con la estructura HTML5, permite detallar como estarán ubicados los elementos en la página principal, los elementos que contiene la aplicación son: un papel tapiz, una sección para el chat, una sección para la transferencia de archivos, un panel para observar al contacto remoto, y una sección para desplegar el video local.

A continuación en la figura 3.32 se detalla la estructura de la página html:

```
<!DOCTYPE html>
<html lang="es">
<head>
    <title></title>
    <link rel="stylesheet" type="text/css" href="">
    <script type="text/javascript"></script>
    <script type="text/javascript"></script>
</head>
<body >
    <header></header>
    <section>
        <div>PARTNERS</div>
        <div>CHAT
            <div>MENSAJES</div>
            <div>ARCHIVOS</div>
        </div>
        <div>LOCAL</div>
    </section>
    <footer></footer>
</body>
</html>
```

Figura 3.32 Estructura html del módulo de comunicación

La instrucción <!DOCTYPE html>, permite conocer al navegador web la versión de HTML con la que está escrita la página, esta instrucción debe ir antes de la etiqueta <html>.

En la cabecera de la página, se coloca el título, la referencia de los *scripts* y librerías que utilizará el módulo y de igual forma los estilos aplicados a la página web. A continuación se define el cuerpo de la página, la cual contiene una cabecera, una sección y un pie o *footer*. Básicamente en el <*body*> se define la estructura y distribución de los elementos de la página html.

La etiqueta <*head*> contiene las referencias detalladas en la figura 3.33:

```
<head>
    <link href="css/bootstrap.min.css" rel="stylesheet" />
    <link href="css/estiloTesis2.css" rel="stylesheet" />
    <script src="js/jquery.min.js"></script>
    <script src="js/bootstrap.min.js"></script>
    <script src="js/XSockets.latest.js"></script>
    <script src="js/XSockets.WebRTC.latest.js"></script>
</head>
```

Figura 3.33 Referencias a estilos y librerías de la página HTML

Las referencias permiten utilizar archivos y librerías que implementa el módulo de comunicación, para darle estilos a la página, realizar el *responsive* de la aplicación y utilizar las API de WebRTC.

El cuerpo de la página web contiene dos ventanas, esto gracias al atributo *z-index* declarado en el archivo *estiloTesis2.css*, que permite observar primero una ventana y luego la siguiente.

Estos atributos son fácilmente accesibles mediante el uso de las herramientas para proporcionar estilos a una página web, *Bootstrap* también ayuda a que todos los elementos que conforman la aplicación, cambien dinámicamente de posición.

Al ingresar al módulo de comunicación aparece una primera ventana (figura 3.34), la cual permite compartir un enlace para que dos usuarios de la aplicación estén en la misma sala de video-llamada de tal forma que puedan intercambiar mensajes y

archivos.

Figura 3.34 Primera ventana del módulo de comunicación

El enlace puede ser compartido por cualquier red social, se ha colocado dos enlaces a las principales redes, uno para Twitter y otro para Facebook, si ya se ha recibido el enlace, simplemente se puede cerrar la ventana.

3.1.4.4.2 estiloTesis2.css

Este archivo permite organizar la presentación y el aspecto de la página web por ejemplo, se puede cambiar el tipo de letra, cambiar colores, posiciones, etc. Para poder referenciar la hoja de estilos se incluye la siguiente instrucción dentro de la etiqueta <*head*>. La estructura de este archivo se lo puede observar en la figura 3.35. <link href="estiloTesis2.css" rel="stylesheet" />

```
header {
    /*properties*/
}
body{
    /*image*/
    /*properties*/
}
#partnerVideo {
    /*properties*/
}
#chatMesssages {
    /*properties*/
}
footer {
    /*properties*/
}
#ventanaM {
    /*properties*/
}
```

Figura 3.35 Estructura del archivo CSS

3.1.4.4.3 Librerías XSockets.latest.js y XSockets.WebRTC.latest.js

Estas librerías se incluyen en el archivo *index*.html mediante la siguiente instrucción:

```
<script src="//../../Scripts/XSockets.latests.js"> </script>

<script src="//../../Scripts/XSockets.WebRTC.latests.js"> </script>
```

Estos *scripts* permiten utilizar instrucciones de XSockets para que el programa las pueda interpretar y no genere error en la implementación del código.

La librería *XSockets.latest.js*, permite desarrollar el *script* que le da la funcionalidad al módulo de comunicación y efectúa tareas de JavaScript con menor esfuerzo y con pocas líneas de código. De esta manera, permite realizar las tareas en un tiempo muy corto.

Las funciones utilizadas en el *script* del archivo *index.html*, se muestran en la tabla 3.4

Tabla 3.4 Funciones utilizadas de la librería XSockets.WebRTC.latest.js

FUNCIONES	DESCRIPCIÓN
XSockets.Utils.guid()	Permite crear un contexto único, basado en el identificador único global.
XSockets.WebSocket()	Conexión con un extremo, el cual recibe una dirección IP y un controlador.
XSockets.WebRTC()	Recibe una instancia de controlador
XSockets.WebRTC.DataChannel()	Crea un canal de datos WebRTC

3.1.4.5 Implementación de Información Complementaria

Para la implementación de información del tipo complementaria, únicamente se debe utilizar el controlador y las vistas.

Debido a que no se deben enviar datos para que sean almacenados, simplemente se mostrará información respecto al uso de la aplicación.

3.1.4.5.1 Controlador

El controlador procesa las solicitudes entrantes, controlan los datos proporcionados por los usuarios y ejecutan la lógica de la aplicación adecuada.

El controlador responde a la solicitud hecha por el usuario, a continuación se define el método de acción que recibe el controlador. Existen dos métodos que se ejecutan, el método *get*, que obtiene los datos proporcionados a la aplicación y el método *post*, el cual realiza la tarea pedida.

El método de acción *GET* se produce una vez que una persona le da *click* a los botones de “Acerca de” o “Contacto” en la página de inicio, lo que provoca que el controlador llame a la vista para permitir que los usuarios puedan informarse acerca del uso de la aplicación y del desarrollador del mismo, la implementación de estos dos métodos se los puede observar en la figura 3.36:

```
0 referencias
public class HomeController : Controller
{
    0 referencias
    public ActionResult Index()
    {
        return View();
    }

    0 referencias
    public ActionResult About()
    {
        return View();
    }

    0 referencias
    public ActionResult Contact()
    {
        return View();
    }
}
```

Figura 3.36 *Home Controller*

3.1.4.5.2 Vistas

La vistas Acerca de y Contacto se pueden observar en la figura 3.37 y figura 3.38 respectivamente.

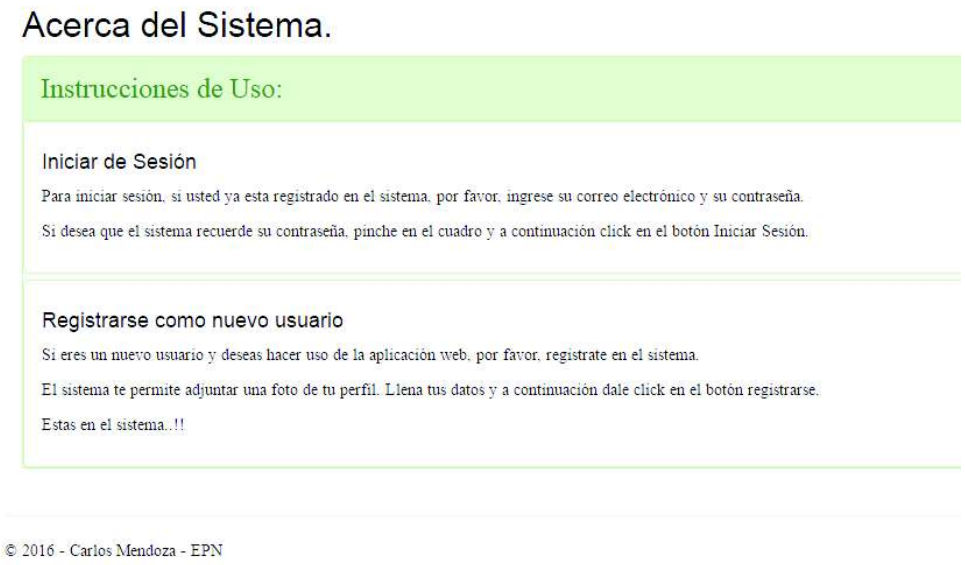


Figura 3.37 Vista acerca del sistema

Figura 3.38 Vista contacto

3.2 PRUEBAS DE FUNCIONAMIENTO

Se ha realizado la verificación de las funcionalidades que la aplicación presenta al usuario, tanto de los módulos que permiten la administración y la autenticación; como del módulo que permite la comunicación en tiempo real entre dos usuarios.

El escenario desplegado para realizar las pruebas, fue el presentado en la figura 3.39:

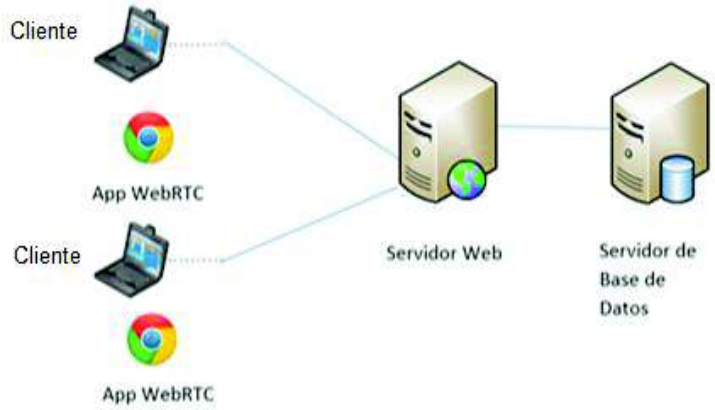


Figura 3.39 Escenario de pruebas

A continuación, en la tabla 3.5 se detallan las características en *hardware* de los equipos usados para realizar las pruebas:

Tabla 3.5 Características de *hardware*

Dispositivos	**Características**
Servidor	Marca: Lenovo Procesador: *CORE 2 Duo* Memoria RAM: 3 GB
Cliente uno	Marca: Toshiba Procesador: *Core* i3 Memoria RAM: 4 GB
Cliente dos	Marca: Lenovo Procesador: AMD 2 Memoria RAM: 4 GB

En la tabla 3.6, se puede observar las características en *software* de los equipos utilizados en el escenario de pruebas:

Tabla 3.6 Características de *software*

Dispositivos	**Características**
Servidor	OS: Windows 8.1 Servidor Web: IIS *Express*
Cliente uno	OS: Windows 8.1 Navegador: Google Chrome
Cliente dos	OS: Windows 8.1 Navegador: Google Chrome

- **Pruebas realizadas:**

Se realizaron las pruebas de funcionalidad considerando cada una de las historias de usuario, para verificar el correcto cumplimiento de dichas historias. En total se tiene un total once historias de usuario que componen todos los módulos de la aplicación. A continuación se presenta las pruebas realizadas:

3.2.1 AUTENTICACIÓN EN LA APLICACIÓN

Esta prueba se realizó para verificar el correcto funcionamiento de la historia de usuario HU5, que valida el ingreso a la aplicación de usuarios y administradores.

En la figura 3.40, se muestran los mensajes de error cuando un usuario, no ingresa información (figura 3.40 (a)), cuando la dirección de correo no es válida (figura 3.40 (b)), o cuando el correo o la contraseña son incorrectos (figura 3.40 (c)).

Figura 3.40 (a)

Figura 3.40 (b)

Figura 3.40 (c)

Figura 3.40 Autenticación de la aplicación - errores

3.2.2 ADMINISTRACIÓN DE USUARIOS

Esta prueba se la realizó para verificar la correcta funcionalidad de la historia de usuario HU1. Como ejemplo se realizó la creación, modificación y correspondiente eliminación de un usuario de la aplicación, en la figura 3.41 se muestran las acciones descritas:

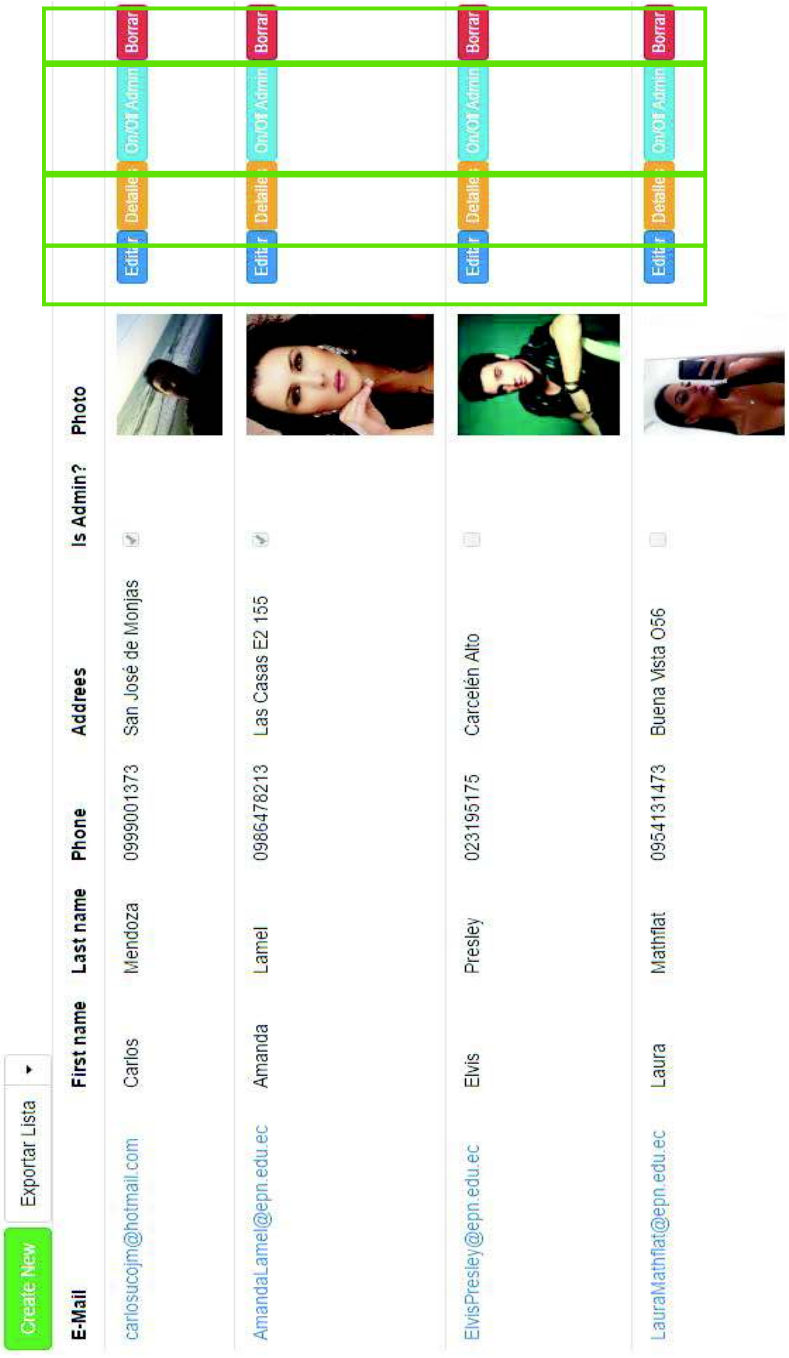


Figura 3.41 Funcionalidades de la administración de usuarios

En la figura anterior, se muestra la vista para acciones CRUD y adicionalmente la acción para cambiar de rol a un usuario dentro de la aplicación.

En la figura 3.42 se presentan las acciones CRUD que tiene a disposición el administrador de la aplicación sobre los usuarios. (*Create* caso (a), *Update* caso (b), *Read* caso (c) y *Delete* caso (d)).

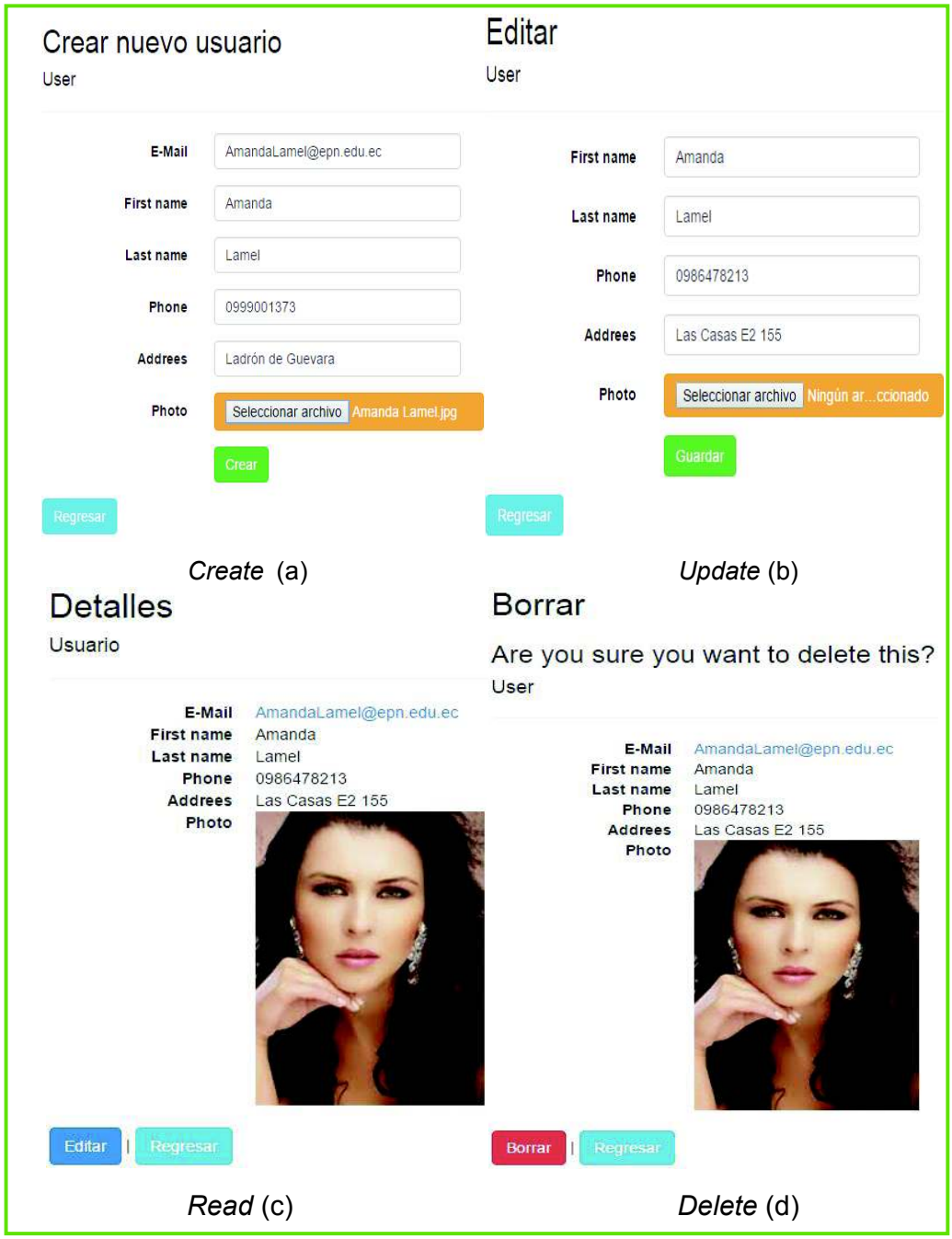


Figura 3.42 Acciones CRUD

3.2.3 ADMINISTRACIÓN DE ADMINISTRADORES

Esta prueba se la realizó, con el fin de verificar la correcta funcionalidad de la historia de usuario HU 2. Se realizó las acciones de registro, modificación y correspondiente eliminación de administradores. La vista es similar a la de la figura 3.42. Adicionalmente se ha cambiado el rol de un usuario, los resultados obtenidos se muestran en la figura 3.43.

Figura 3.43 Cinta de opciones de la aplicación

3.2.4 ADMINISTRACIÓN DE GRUPOS

Las siguientes pruebas fueron realizadas para verificar el correcto funcionamiento de las historias de usuario HU 3 y HU 4. El administrador tiene opciones CRUD sobre los grupos de la aplicación y tiene la posibilidad de agregar o eliminar miembros de un grupo determinado. A continuación, en la figura 3.44 se presentan los métodos mencionados.

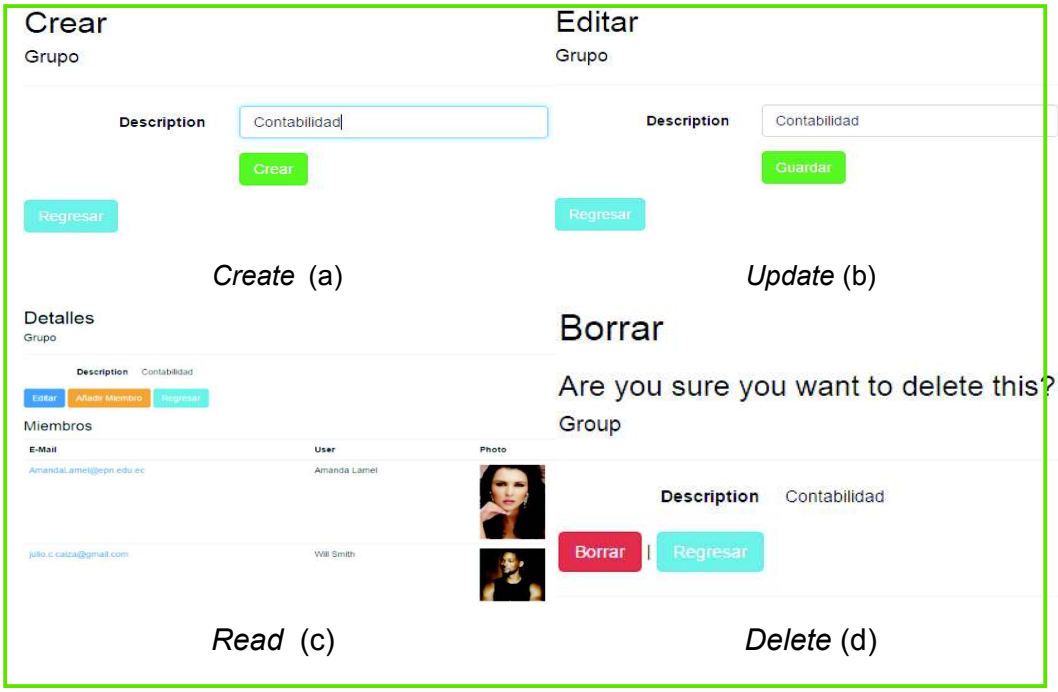


Figura 3.44 Acciones CRUD grupos

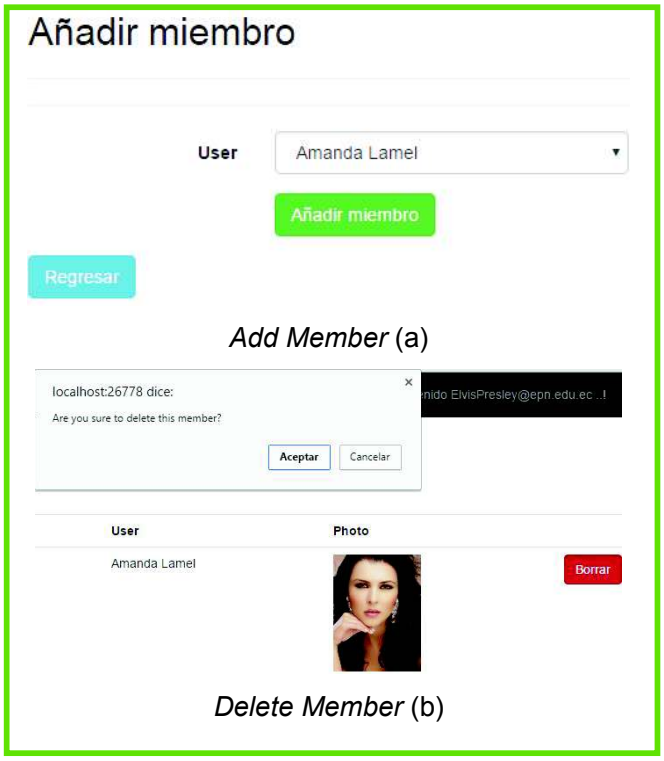


Figura 3.45 Acciones sobre miembros de grupo

En la figura 3.45, se muestra las opciones que tiene un administrador para agregar miembros a un grupo, con lo cual se verificó el correcto funcionamiento de la historia de usuario HU 4.

3.2.5 VIDEO LLAMADAS, TRANSFERENCIA DE CHAT Y ARCHIVOS

Las siguientes pruebas fueron realizadas para verificar el correcto funcionamiento del módulo de comunicación, compuesto por los elementos descritos en la sección anterior y comprobar las historias de usuario HU 6, HU 7 y HU 8 detallas en la sección de análisis de requerimientos.

En las cuales un usuario podrá comunicarse en tiempo real con otro usuario registrado en el sistema.

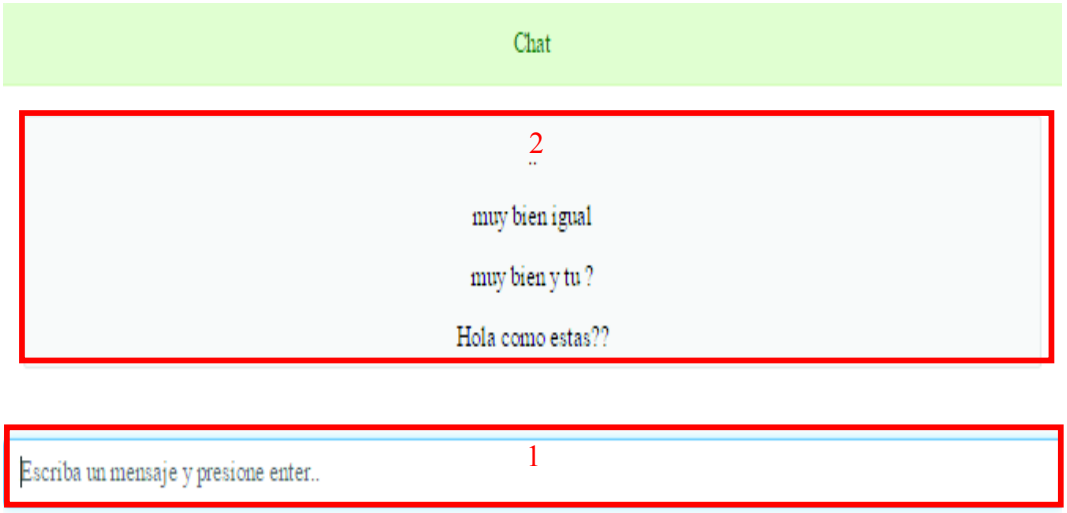
En la figura 3.46 se puede observar el establecimiento de la video llamada entre dos usuarios, estos se encuentran en la misma Intranet, de esta forma, únicamente se debe configurar adecuadamente el *firewall* de cada *host* en dicha red.

Figura 3.46 Video-llamada

A continuación, se presenta las pruebas de funcionamiento para el intercambio de mensajes (HU 7), entre dos usuarios, que permite comprobar el correcto funcionamiento de la historia de usuario 7.

La sección de chat se divide en dos partes, una que permite la redacción del mensaje, el cual se envía presionando la tecla *Enter*; y la otra que permite la visualización de dichos mensajes.

La longitud de los caracteres que componen los mensajes puede ser de hasta 500 caracteres,



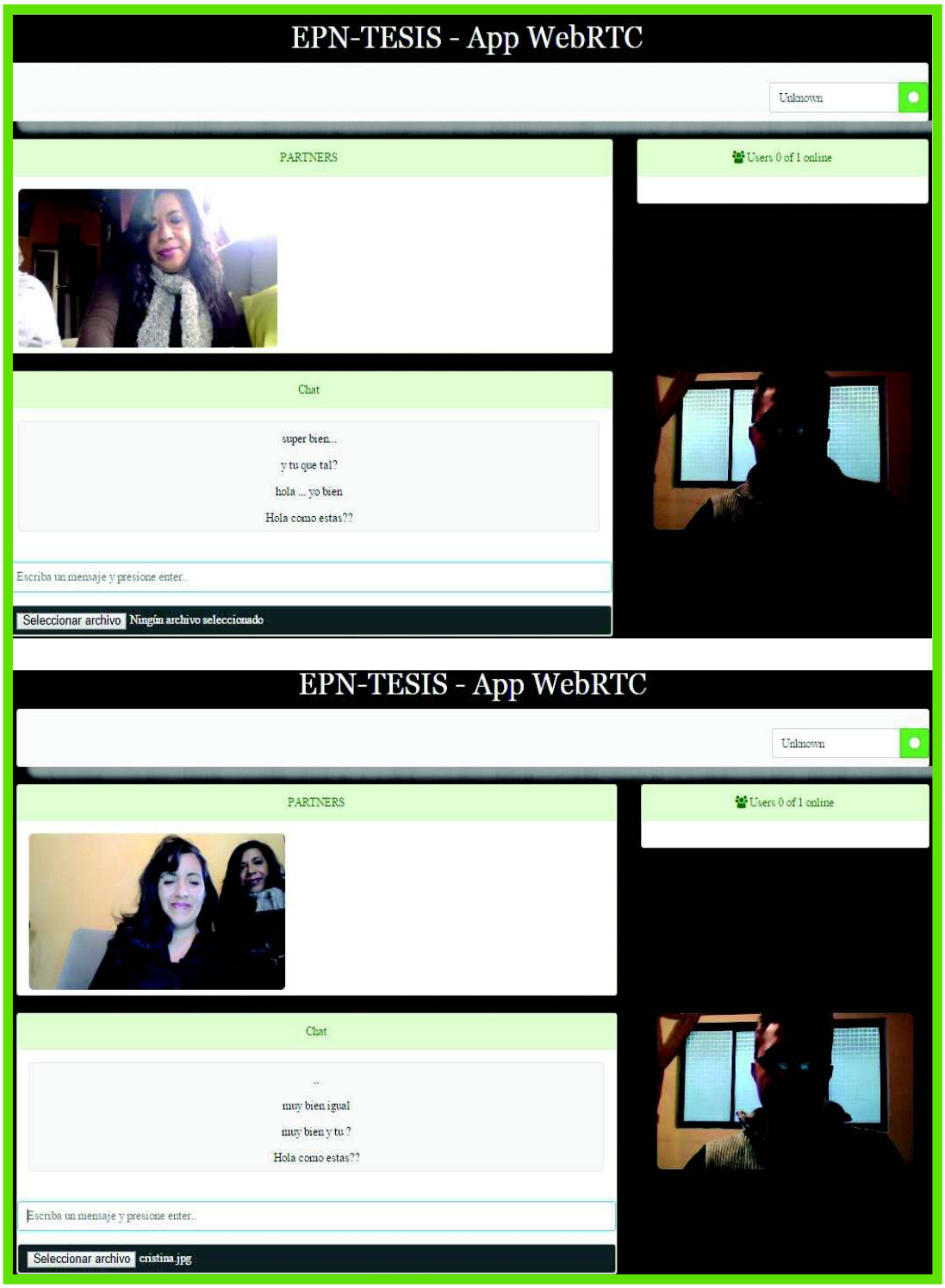


Figura 3.47 Intercambio de Mensajes

Finalmente se realizaron las pruebas con la transferencia de archivos, que básicamente han sido de dos tipos, imágenes y textos. A continuación se presenta en la figura 3.48 el ejemplo de un archivo enviado desde un usuario a otro:

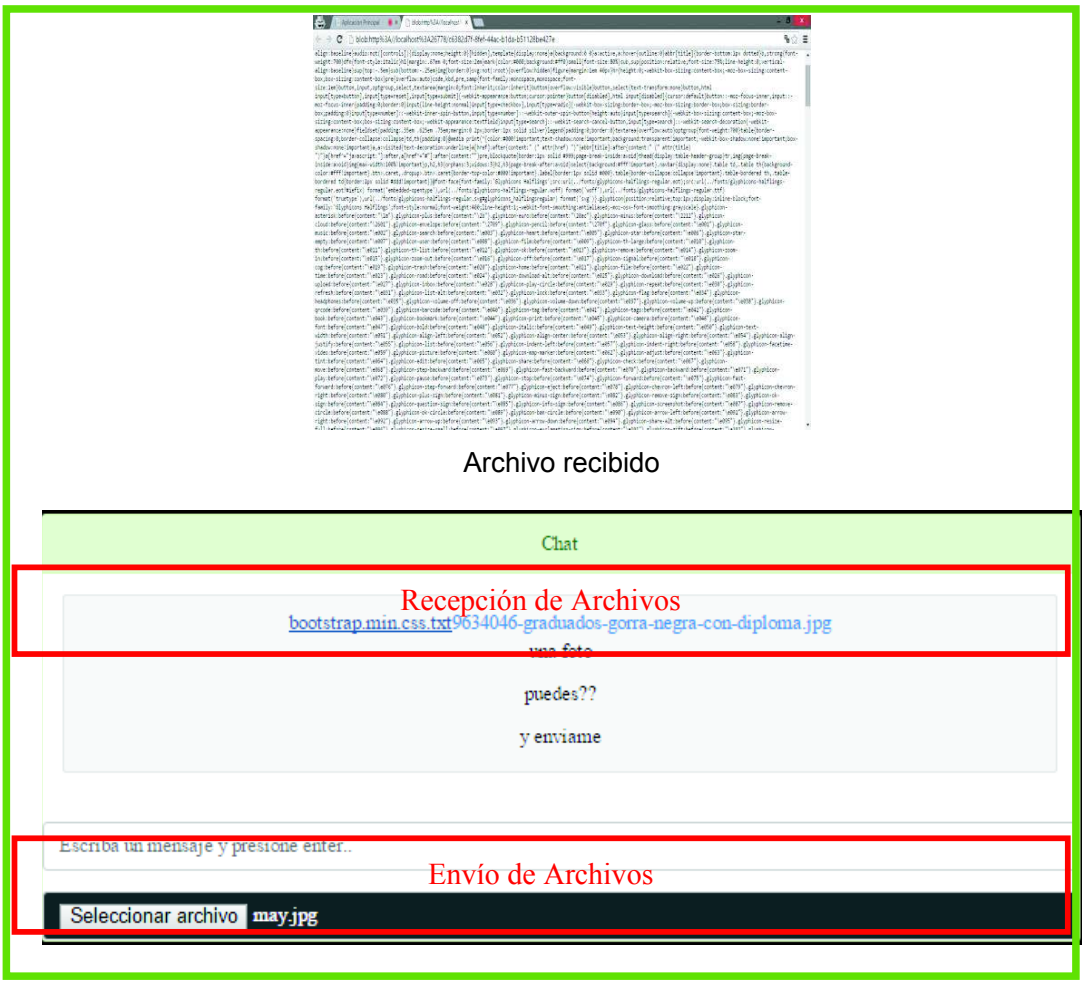


Figura 3.48 Intercambio de archivos

4 CAPÍTULO 4: CONCLUSIONES Y RECOMENDACIONES

En este capítulo se presentan las conclusiones y las recomendaciones obtenidas a lo largo de la elaboración de este proyecto.

4.1 CONCLUSIONES

- Trabajar con XSockets y WebRTC ha permitido desarrollar una aplicación web transparente al sistema operativo en el cual se ejecuta, ya que, únicamente se debe tener en cuenta la compatibilidad con el navegador y la versión del *framework* .NET (4.0 +) utilizados.

- WebRTC al tener sus componentes implementados en el navegador, permitió que estos sean accesibles fácilmente mediante el uso de las API JavaScript, reduciendo en gran medida la complejidad del desarrollo del módulo de comunicación. Es decir, para un desarrollador web le resultaría muy fácil integrar comunicaciones en tiempo real dentro de sus aplicaciones.

- El uso del patrón MVC ha logrado dividir las diferentes partes que conforman la aplicación web, permitiendo una actualización y mantenimiento de *software* de forma sencilla y en un espacio reducido de tiempo.

- El uso del *framework* .NET en conjunto con el uso de las plantillas MVC para el desarrollo de este proyecto, ha permitido una separación lógica de los componentes de la aplicación web, lo que ha contribuido a una mejor organización en el trabajo.

- El sistema gestor de base de datos SQL Server utilizado, ha permitido un mejor rendimiento en el manejo de los tiempos en las consultas realizadas por parte de los usuarios de la aplicación web.

- Utilizar *Bootstrap* para el desarrollo permitió adaptar automáticamente la aplicación web al tamaño de pantalla en el que esta se visualice. Permitiendo así un diseño adaptivo, simple e intuitivo y reduciendo la carga de trabajo.

- De igual manera, el uso de *Bootstrap* en conjunto con CSS3 permitió desarrollar una aplicación web con una muy buena presentación visual, agregando estilos para cada uno de los componentes que lo conforman.

- El uso de las Anotaciones de Datos simplificó en gran medida el manejo de validaciones en el lado del cliente, de esta forma se pudo desplegar en cada vista que propiedades fueron requeridas o necesarias y generar dinámicamente un mensaje con los nombres de propiedad y de anotación.

- El módulo de administración de grupos, reúne de forma ordenada equipos de usuarios que pueden pertenecer a determinadas áreas de trabajo, lo que permitió utilizar los recursos de comunicación en tiempo real de forma segmentada.

- *Entity Framework* permitió desarrollar la aplicación web con un modelo de aplicaciones conceptual, en lugar de trabajar con un esquema de almacenamiento relacional, esto ayudó a reducir la cantidad de líneas de código implementadas y proporcionó una mayor libertad de dependencia de codificación rígida de un motor de datos.

4.2 RECOMENDACIONES

- Se recomienda trabajar con el navegador Google Chrome, debido a que Google ha sido el principal motor de WebRTC, con lo cual proporciona una completa integración y soporte básico a través de API JavaScript. Otro navegador en el que se ha realizado las pruebas fue Mozilla Firefox.

- Para trabajos futuros sobre este tema se recomienda tener conocimientos avanzados en JavaScript, de esta forma se podrían personalizar las API de WebRTC para que realicen determinadas funciones, en un espacio reducido de tiempo.

- Un inconveniente que se ha presentado es que WebRTC aún está en proceso de estandarización, por lo cual no ha definido un método de señalización. Se recomienda utilizar SIP debido a que proporciona mayores facilidades en la implementación y existe mayor cantidad de información para una etapa de investigación.

- WebRTC promete ser una tecnología disruptiva, ya que muchas operadoras de telecomunicaciones (AT&T, Orange, Telefónica, entre otras), investigan las diversas oportunidades que esta ofrece. Es muy importante continuar con este tipo de investigaciones que permitirían reducir costes y mejorar la experiencia como usuarios de servicios de telecomunicaciones.

REFERENCIAS BIBLIOGRÁFICAS

[1] S. Loreto y S. Prieto, Real Time Communication with WebRTC, Sebastopol CA: O'REILLY, 2014.

[2] K. Geek, «WebRTC,» 2012. [En línea]. Available: https://webrtc.org/start/.

[3] T. Levent-Levi, «What is WebRTC,» 8 March 2012. [En línea]. Available: https://bloggeek.me/webrtc/.

[4] Anónimo, «Global IP Solutions,» 11 August 2015. [En línea]. Available: https://en.wikipedia.org/wiki/Global_IP_Solutions.

[5] S. Dutton, «Getting Started with WebRTC,» 23 July 2012. [En línea]. Available: http://www.html5rocks.com/en/tutorials/webrtc/basics/#toc-simple.

[6] R. Millán, «WebRTC: comunicaciones en tiempo real en el navegador Web,» 2014. [En línea]. Available: http://www.ramonmillan.com/tutoriales/webrealtimecommunications.php.

[7] J. Gauchat, El gran libro de HTML5, CSS3 y JavaScript, marcombo.

[8] M. Alvarez, «Desarrolladores web,» Junio 2012. [En línea]. Available: http://www.desarrolloweb.com/articulos/etiquetas-semanticas-html5.html.

[9] Anónimo, «The HTML 5 JavaScript API,» [En línea]. Available: http://html5index.org/.

[10] U. Björklund, «XSockets Targeted Real-Time Event Distribution,» 2010. [En línea]. Available: https://uffebjorklund.gitbooks.io/xsockets-net-5/content/index.html.

[11] Anónimo, «Patrón Modelo-Vista-Controlador,» [En línea]. Available: http://www.proactiva-calidad.com/java/patrones/mvc.html.

[12] E. Catalani, «Arquitectura Modelo/Vista/Controlador,» 2007. [En línea]. Available: http://exequielc.wordpress.com/2007/08/20/arquitectura.

[13] Neleste, «Modelo Vista Controlador,» 22 Julio 2008. [En línea]. Available: http://www.neleste.com/modelo%20-%20vista%20-%20controlador/.

[14] Anónimo, «El patrón Modelo-Vista-Controlador,» [En línea]. Available: https://www.fdi.ucm.es/profesor/jpavon/poo/2.14.MVC.pdf.

[15] J. Figueroa, «*Framework*s MVC de desarrollo Web,» 2008. [En línea]. Available: http://blog.buhoz.net/blog1.php/2008/03/06/*framework*s.

[16] T. Dykstra, «Creating ASP.NET Web Projects,» 1 December 2014. [En línea]. Available: http://www.asp.net/visual-studio/overview/2013/creating-web-projects-in-visual-studio#1684.

[17] P. Gasston, «The modern Web,» 2013. [En línea]. Available: http://proquest.safaribooksonline.com.

[18] Microsoft, «Entity *Framework*,» [En línea]. Available: http://www.asp.net/entity-*framework*.

[19] Microsoft, «System.Web.Security,» [En línea]. Available: https://msdn.microsoft.com/es-es/library/system.web.security(v=vs.110).aspx.

[20] Microsoft, «Controladores y Métodos de acción en aplicaciones ASP.NET MVC,» [En línea]. Available: https://msdn.microsoft.com/es-ec/library/dd410269(v=vs.100).aspx.

[21] Microsoft, «Introducción a Visual Studio,» [En línea]. Available: https://msdn.microsoft.com/es-es/library/fx6bk1f4(v=vs.90).aspx.

[22] E. Sojo, «Usando Nuget,» 2011. [En línea]. Available: http://eduardosojo.com/2011/07/19/usuando-nuget-que-es-nuget/.

[23] R. Silveira, «Learn HTML5,» 2013. [En línea]. Available: http://proquest.safaribooksonline.com.

[24] P. Salinas, «Modelo de Clases,» [En línea]. Available: http://users.dcc.uchile.cl/~psalinas/uml/modelo.html.

[25] P. Project, «open-source GUI prototyping tool,» [En línea]. Available: http://pencil.evolus.vn/Default.html.

[26] Microsoft, «Using Data Annotations,» [En línea]. Available: https://msdn.microsoft.com/en-us/library/dd901590(VS.95).aspx.

[27] Linux, «Introducción a la administración de sistemas,» [En línea]. Available: http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-isa-es-4/ch-acctsgrps.html.

[28] A. Jhonston y D. Burnett, WebRTC: The Web Way to Communicate, St. Louis: COMSOC, 2013.

[29] S. Dutton, «WebRTC Plugin-free real-time communication,» *Google Chrome Developer Relations,* p. 59, 2012.

[30] P. Caro, «Metodología Gestión de Requerimientos,» 2012. [En línea]. Available: https://sites.google.com/site/metodologiareq/.

[31] Anónimo, "Introducción a Boostrap". [En línea]. Available: http://getbootstrap.com/css/.

ANEXOS

ANEXO A: MANUAL DEL USUARIO

ANEXO B: ANOTACIONES DE DATOS

ANEXO C: CÓDIGO DE LA APLICACIÓN WEB

ANEXO D: PRUEBAS DE FUNCIONAMIENTO

Nota: Los anexos que se enumeran a continuación se encuentran en el CD adjunto.