

**Tema**

SOCKETS DOTNET

**Tutor**

Ing. Eduardo Mauricio Campaña Ortega

MIS. MDU.CCNA. CCIA.

PhD. (c) Ingeniería de Software

PhD. (c) Seguridad Información

**Fecha**

18/01/2023

SOCKETS EN DOTNET

[1. INTRODUCCIÓN 5](#_Toc124849795)

[2. OBJETIVOS 5](#_Toc124849796)

[2.1. OBJETIVO GENERAL 5](#_Toc124849797)

[2.2. OBJETIVOS ESPECÍFICOS 5](#_Toc124849798)

[3. MARCO TEÓRICO 5](#_Toc124849799)

[3.1 ARQUITECTURA CLIENTE – SERVIDOR 5](#_Toc124849800)

[3.1.1 Cliente 6](#_Toc124849801)

[3.1.2 Servidor 6](#_Toc124849802)

[3.2 SOCKETS 7](#_Toc124849803)

[3.2.1 PROPIEDADES 8](#_Toc124849804)

[3.2.2 ATRIBUTOS 8](#_Toc124849805)

[3.2.3 TIPOS DE SOCKETS 8](#_Toc124849806)

[3.3 APLICACIÓN DE CONSOLA C# 9](#_Toc124849807)

[3.4 APLICACIÓN DE WPF (.NET FRAMEWORK) 9](#_Toc124849808)

[3.5 ARCHIVOS EN WPF (.NET FRAMEWORK) 10](#_Toc124849809)

[3.5.1 XAMARIN FORMS 10](#_Toc124849810)

[3.5.2 ARCHIVO DE RECURSOS .RESX 16](#_Toc124849811)

[3.6 Serialización y Deserialización 17](#_Toc124849812)

[3.6.1 Serialización y Deserialización con BinaryFormatter 18](#_Toc124849813)

[3.7 Espacio de nombres 18](#_Toc124849814)

[4. PARTE PRÁCTICA 19](#_Toc124849815)

[4.1 CREACIÓN DE CARPETAS PARA EL PROYECTO 19](#_Toc124849816)

[4.2 PROYECTO SERVIDOR 19](#_Toc124849817)

[4.2.1 CREACIÓN DEL PROYECTO SERVIDOR 19](#_Toc124849818)

[4.2.2 ESTRUCTURA DEL PROYECTO SERVIDOR (MODELO-VISTA-CONTROLADOR) 21](#_Toc124849819)

[4.2.3 CREACIÓN DE ARCHIVOS PARA CADA CARPETA 22](#_Toc124849820)

[4.2.4 CODIFICACIÓN DEL PROYECTO SERVIDOR 25](#_Toc124849821)

[4.3 PROYECTO CLIENTE 30](#_Toc124849822)

[4.3.1 CREACIÓN DEL PROYECTO CLIENTE 30](#_Toc124849823)

[4.3.2 ESTRUCTURA DEL PROYECTO CLIENTE (MODELO-VISTA-CONTROLADOR) 32](#_Toc124849824)

[4.3.3 CREACIÓN DE ARCHIVOS PARA CADA CARPETA 33](#_Toc124849825)

[4.3.4 CODIFICACIÓN DEL PROYECTO CLIENTE 42](#_Toc124849826)

[4.1 EJECUCIÓN DEL PROYECTO 56](#_Toc124849827)

[4 CONCLUSIONES 63](#_Toc124849828)

[5 RECOMENDACIONES 63](#_Toc124849829)

[6 REFERENCIAS 64](#_Toc124849830)

**INDICE DE IMÁGENES**

[Figura 1. Arquitectura Cliente – Servidor. 8](#_Toc124850898)

[Figura 2. Funcionamiento base de Sockets. 9](#_Toc124850899)

[Figura 3. Comunicación entre dos sockets. 9](#_Toc124850900)

[Figura 4. Sockets 10](#_Toc124850901)

[Figura 5. Aplicación de consola C#. 10](#_Toc124850902)

[Figura 6. Aplicación de WPF (.NET Framework) 11](#_Toc124850903)

[Figura 7. Ejemplo de un archivo XAML. 11](#_Toc124850904)

[Figura 8. Anatomía de un archivo xaml. 12](#_Toc124850905)

[Figura 9. Archivos de Recursos 18](#_Toc124850906)

[Figura 10. Tipos de archivos 18](#_Toc124850907)

[Figura 11. Serialización y deserialización. 19](#_Toc124850908)

[Figura 12. Estructura del proyecto. 20](#_Toc124850909)

[Figura 13. Pantalla principal de Visual Studio 2022. 21](#_Toc124850910)

[Figura 14. Selección del tipo de proyecto. 21](#_Toc124850911)

[Figura 15. Configuración del proyecto Servidor. 22](#_Toc124850912)

[Figura 16. Estructura de un proyecto Aplicación de consola. 22](#_Toc124850913)

[Figura 17. Creación de carpetas Modelo, Vista, Controlador 23](#_Toc124850914)

[Figura 18. Estructura del proyecto. 23](#_Toc124850915)

[Figura 19. Creación de archivos en la carpeta Modelo. 24](#_Toc124850916)

[Figura 20. Selección de tipo de archivo y nombre. 24](#_Toc124850917)

[Figura 21. Creación de archivos en la carpeta Vista. 24](#_Toc124850918)

[Figura 22. Selección de tipo de archivo y nombre. 25](#_Toc124850919)

[Figura 23. Selección del tipo de archivo y asignación de nombre. 25](#_Toc124850920)

[Figura 24. Selección del tipo de archivo y asignación de nombre. 26](#_Toc124850921)

[Figura 25. Estructura del proyecto Servidor, Carpetas y Archivos. 26](#_Toc124850922)

[Figura 26. Pantalla principal de Visual Studio 2022. 32](#_Toc124850923)

[Figura 27. Selección del tipo de aplicación para el proyecto Cliente. 32](#_Toc124850924)

[Figura 28. Configuración de parámetros del proyecto Cliente. 33](#_Toc124850925)

[Figura 29. Estructura del proyecto Cliente. 33](#_Toc124850926)

[Figura 30. Creación de carpetas Modelo, Vista y Controlador. 34](#_Toc124850927)

[Figura 31. Creación de carpetas para el proyecto Cliente. 34](#_Toc124850928)

[Figura 32. Estructura del proyecto Cliente. 35](#_Toc124850929)

[Figura 33. Agregar nuevo elemento a la carpeta Modelo. 35](#_Toc124850930)

[Figura 34. Selección del tipo de archivo y asignación de nombre 36](#_Toc124850931)

[Figura 35. Agregar nuevo elemento a la carpeta Vista. 36](#_Toc124850932)

[Figura 36. Selección del tipo de archivo y asignación de nombre a los archivos. 36](#_Toc124850933)

[Figura 37. Agregar nuevo elemento a la carpeta Controlador. 37](#_Toc124850934)

[Figura 38. Selección del tipo de archivo y asignación de nombre. 37](#_Toc124850935)

[Figura 39. Agregar nuevo elemento a la carpeta Utils. 37](#_Toc124850936)

[Figura 40. Selección del tipo de archivo y asignación de nombre. 38](#_Toc124850937)

[Figura 41. Agregar elementos nuevos a la carpeta Components. 38](#_Toc124850938)

[Figura 42. Selección del tipo de archivo y asignación de nombre. 39](#_Toc124850939)

[Figura 43. Fuente Montserrat 39](#_Toc124850940)

[Figura 44. Fuente Montserrat Alternates 39](#_Toc124850941)

[Figura 45. Archivos descargados. 39](#_Toc124850942)

[Figura 46. Extraer archivos contenidos en las carpetas descargadas. 40](#_Toc124850943)

[Figura 47. Selección del tipo de fuente de cada carpeta. 40](#_Toc124850944)

[Figura 48. Copiar los tipos de fuente en la carpeta Fonts. 40](#_Toc124850945)

[Figura 49. Agregar elemento existente a la carpeta Fonts. 41](#_Toc124850946)

[Figura 50. Selección de archivos que se agregaran a la carpeta Fonts. 41](#_Toc124850947)

[Figura 51. Estructura de la carpeta Fonts. 42](#_Toc124850948)

[Figura 52. Agregar imágenes en la carpeta Resources. 42](#_Toc124850949)

[Figura 53. Agregamos elementos existentes. 42](#_Toc124850950)

[Figura 54. Agregar archivo de recursos. 43](#_Toc124850951)

[Figura 55. Selección del tipo de archivo y asignación de nombre. 43](#_Toc124850952)

[Figura 56. Asignación de recursos al archivo. 43](#_Toc124850953)

[Figura 57. Ejecución del proyecto Servidor 58](#_Toc124850954)

[Figura 58. Servidor esperando la conexión de los usuarios. 59](#_Toc124850955)

[Figura 59. Ejecución del proyecto Cliente 59](#_Toc124850956)

[Figura 60. Vista principal de la aplicación de Chat. 59](#_Toc124850957)

[Figura 61. Ingreso del nombre del primer usuario. 60](#_Toc124850958)

[Figura 62. Ventana principal del Chat del usuario Kevin. 60](#_Toc124850959)

[Figura 63. Ingreso del nombre del segundo usuario. 60](#_Toc124850960)

[Figura 64. Lista de usuarios con los cuales se puede establecer una comunicación. 61](#_Toc124850961)

[Figura 65. Mensaje del usuario Steven a Kevin. 61](#_Toc124850962)

[Figura 66. Comunicación entre los usuarios Steven y Kevin. 62](#_Toc124850963)

[Figura 67. Actividad del cliente. 62](#_Toc124850964)

[Figura 68. Elección de la función archivo. 62](#_Toc124850965)

[Figura 69. Selección del archivo para enviar. 63](#_Toc124850966)

[Figura 70. Notificación de descarga del archivo que Steven envía a Kevin. 63](#_Toc124850967)

[Figura 71. Elección de ruta de descarga del archivo recibido 64](#_Toc124850968)

[Figura 72. Notificación de archivo descargado. 64](#_Toc124850969)

[Figura 73. Visualización del archivo descargado. 64](#_Toc124850970)

[Figura 74. Notificación y envió de mensaje al usuario de Archivo rechazado. 65](#_Toc124850971)

**ÍNDICE DE TABLAS**

[Tabla 1. Vista para presentación. 14](#_Toc124850972)

[Tabla 2. Vista para que inicien comandos. 15](#_Toc124850973)

[Tabla 3. Vista para establecer valores. 16](#_Toc124850974)

[Tabla 4. Vistas para editar texto 16](#_Toc124850975)

[Tabla 5. Vista para indicar actividad 17](#_Toc124850976)

[Tabla 6. Vistas para mostrar colecciones. 17](#_Toc124850977)

[Tabla 7. Codificación del archivo User.cs 27](#_Toc124850978)

[Tabla 8. Codificación del archivo Message.cs 28](#_Toc124850979)

[Tabla 9. Codificación del archivo ViewServer.cs 28](#_Toc124850980)

[Tabla 10. Codificación del archivo Server.cs 29](#_Toc124850981)

[Tabla 11. Codificación del archivo Serialization.cs 31](#_Toc124850982)

[Tabla 12. Codificación del archivo Program.cs 31](#_Toc124850983)

[Tabla 13. Codificación del archivo User.cs 44](#_Toc124850984)

[Tabla 14. Codificación del archivo Message.cs 44](#_Toc124850985)

[Tabla 15. Codificación del archivo Chat.cs 45](#_Toc124850986)

[Tabla 16. Codificación del archivo Client.cs 46](#_Toc124850987)

[Tabla 17. Codificación del archivo Serialization.cs 48](#_Toc124850988)

[Tabla 18. Codificación del archivo Input.xaml 48](#_Toc124850989)

[Tabla 19. Codificación del archivo Input.xaml.cs 49](#_Toc124850990)

[Tabla 20. Codificación del archivo Button.xaml 49](#_Toc124850991)

[Tabla 21. Codificación del archivo Button.xaml.cs 50](#_Toc124850992)

[Tabla 22. Codificación del archivo Inputfile.xaml 50](#_Toc124850993)

[Tabla 23. Codificación del archivo InputFile.xaml.cs 51](#_Toc124850994)

[Tabla 24. Codificación del archivo InputFile.xaml 51](#_Toc124850995)

[Tabla 25. Codificación del archivo Login.xaml.cs 52](#_Toc124850996)

[Tabla 26. Codificación del archivo ChatPage.xaml 52](#_Toc124850997)

[Tabla 27. Codificación del archivo ChatPage.xaml.cs 54](#_Toc124850998)

[Tabla 28. Codificación del archivo MainWindows.xaml 58](#_Toc124850999)

[Tabla 29. Codificación del archivo MainWindows.xaml.cs 58](#_Toc124851000)

# INTRODUCCIÓN

Un socket es una interfaz normalizada de comunicación que se utiliza para conectarse a otros procesos a través de una red. Fue diseñado inicialmente para ser utilizado en la arquitectura TCP/IP, pero también puede ser utilizado con otros protocolos de red.

Un socket se compone de dos extremos, cada uno de los cuales está asociado a un proceso de aplicación que se ejecuta en una máquina en la red. Cada extremo del socket se direcciona mediante la dirección IP o el nombre DNS de la máquina y un número de puerto, que debe ser un número entero comprendido entre 1 y 65535.

Los sockets se utilizan a menudo para permitir la comunicación entre programas que ejecutan en diferentes máquinas, pero también pueden ser utilizados para permitir la comunicación entre procesos que ejecutan en la misma máquina. Por ejemplo, un programa puede utilizar un socket para enviar mensajes a otro programa que se esté ejecutando en el mismo ordenador.

Existen diferentes tipos de sockets, como sockets de flujo, sockets de datagrama y sockets de raw, que se utilizan en diferentes contextos y para diferentes propósitos. Por ejemplo, los sockets de flujo se utilizan a menudo para la transmisión de datos en tiempo real, mientras que los sockets de datagrama se utilizan para la transmisión de paquetes de datos no confiables.

# OBJETIVOS

## OBJETIVO GENERAL

Desarrollar e implementar un chat que permita el envío de mensajes de texto y archivos haciendo uso de Sockets en el lenguaje de programación C#.

## OBJETIVOS ESPECÍFICOS

* Analizar la arquitectura Modelo-Vista-Controlador en ASP.NET Framework para comprender como se debe organizar una aplicación web bajo esta arquitectura.
* Desarrollar e implementar un proyecto Servidor haciendo uso de la arquitectura Modelo-Vista-Controlador.
* Desarrollar e implementar un proyecto Cliente haciendo uso de la arquitectura Modelo-Vista-Controlador los cuales se conectan a una misma conversación.
* Permitir la comunicación entre dos o más clientes de tal manera que se puedan enviar mensajes de texto y archivos (texto, imágenes, audios) entre los usuarios que se encuentren en línea.

# MARCO TEÓRICO

## ARQUITECTURA CLIENTE – SERVIDOR

El modelo o arquitectura Cliente – Servidor comenzó a ser aceptado a finales de los 80’s. El funcionamiento o su implementación es sencillo ya que se tiene una máquina cliente la cual requiere de un servicio que se encuentra en una máquina servidor.

Desde el punto de vista funcional, se puede definir la computación Cliente/Servidor como una arquitectura distribuida que permite a los usuarios finales obtener acceso a la información en forma transparente aún en entornos multiplataforma.

Cuando se habla de la arquitectura cliente – servidor, el cliente envía un mensaje realizando una solicitud de cualquiera de los servicios del servidor lo cual se conoce como una petición, y el servidor responde enviando uno o varios mensajes al cliente.

Al ser considerada una arquitectura distribuida se puede decir que las máquinas clientes puede ser también servidores dependiendo de la necesidad, es decir cumplen el rol de cliente y servidor.

La idea es tratar a una computadora como un instrumento, que por sí sola pueda realizar muchas tareas, pero con la consideración de que realice aquellas que son más adecuadas a sus características.

### **Cliente**

El cliente es el proceso que permite al usuario formular los requerimientos y pasarlos al servidor, se le conoce con el término front-end.

El objetivo principal del cliente es la manipulación y el despliegue de los datos, y tiene como característica el desarrollo sobre interfaces gráficas de usuario (GUI).

Procesos de un cliente:

* Administrar la interfaz de usuario.
* Interactuar con el usuario.
* Procesar la lógica de la aplicación y hacer validaciones locales.
* Generar requerimientos de bases de datos.
* Recibir resultados del servidor.
* Formatear resultados.

### **Servidor**

Es el proceso encargado de atender a múltiples clientes que hacen peticiones de algún recurso administrado por él. Al proceso servidor se le conoce con el término back-end.

El objetivo principal del servidor es el manejar y gestionar las reglas del negocio, además, de los recursos de los datos.

Procesos de un servidor:

* Aceptar los requerimientos de bases de datos que hacen los clientes.
* Procesar requerimientos de bases de datos.
* Formatear datos para transmitirlos a los clientes.
* Procesar la lógica de la aplicación y realizar validaciones a nivel de bases de datos.

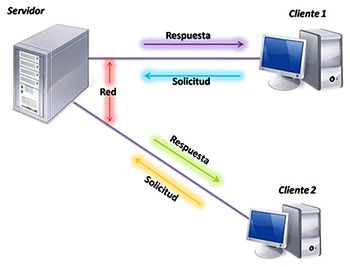


Figura 1. Arquitectura Cliente – Servidor.

## SOCKETS

Un socket es un mecanismo de comunicación que permite a los procesos interactuar entre sí, ya sea en la misma máquina o en distintas máquinas conectadas en una red. Se trata de un punto final de comunicación en el sistema operativo que permite a los procesos enviar y recibir información a través de una conexión de red.

Los sockets se utilizan en una amplia variedad de aplicaciones, desde la comunicación de red básica hasta aplicaciones en tiempo real, como los juegos en línea. Además, los sockets también son utilizados en sistemas distribuidos, como los sistemas de bases de datos distribuidas o los sistemas de nube.

Existen diferentes tipos de sockets, cada uno de ellos diseñado para adaptarse a diferentes necesidades de comunicación. Los sockets de conexión son utilizados para aplicaciones en las que los procesos necesitan establecer una conexión previa antes de intercambiar información. Los sockets de conexión orientados a conexión, por otro lado, son utilizados para aplicaciones en las que los procesos necesitan establecer una conexión previa y mantenerla mientras intercambian información. Los sockets no orientados a conexión, por último, son utilizados para aplicaciones en las que los procesos no necesitan establecer una conexión previa antes de intercambiar información.

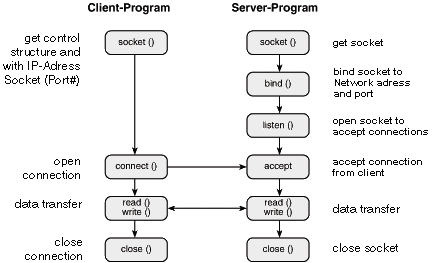


Figura 2. Funcionamiento base de Sockets.

### **PROPIEDADES**

* Fiabilidad de la transmisión, los datos que transmiten no se pierden.
* Conservación del orden de los datos, los datos mantienen el orden de llegada tal como fueron emitidos.
* No existen datos duplicados, los datos llegan una sola vez a su destino.
* Comunicación en modo conectado, la conexión se encuentra establecida antes de comenzar con la comunicación.
* Conservación de los límites de los lenguajes, los límites de mensajes emitidos pueden encontrarse o conocerse en el destino.
* Envío de mensajes urgentes, permite el envío de mensajes fuera del flujo cuando se envía un mensaje fuera del flujo.

### **ATRIBUTOS**

* Dominio: especifica el medio de comunicación de la red que el socket utilizará.
* Protocolo: especifica que protocolo se va a utilizar.
* Tipo: los protocolos de internet proveen dos niveles distintos de servicio, flujo y datagramas.

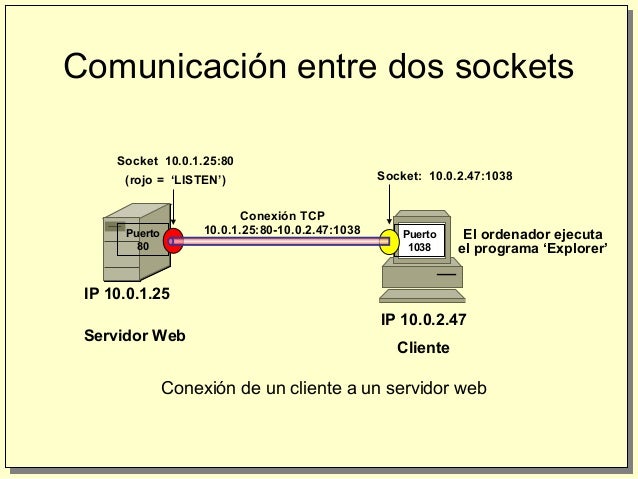


Figura 3. Comunicación entre dos sockets.

### **TIPOS DE SOCKETS**

* Orientados a conexión: contiene comunicaciones fiables y circuitos virtuales.
* No orientados a conexión: el programa de aplicación da la fiabilidad.

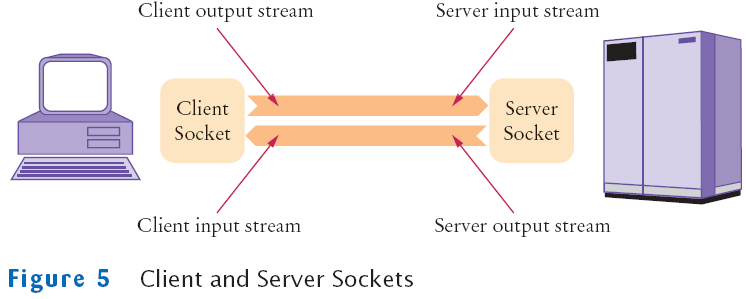


Figura 4. Sockets

## APLICACIÓN DE CONSOLA C#

Una aplicación de consola es un tipo de programa informático que se ejecuta en una consola o terminal de texto. Esto significa que la interfaz del usuario se basa en texto y no en gráficos. Las aplicaciones de consola son comunes en sistemas operativos como Linux y MacOS, y también se pueden crear en Windows utilizando el lenguaje de programación C#.

Para crear una aplicación de consola en C#, primero se requiere instalar un entorno de desarrollo de C#, como Visual Studio o MonoDevelop. Luego, se deberá crear un nuevo proyecto de aplicación de consola y escribirás el código C# en el archivo principal del proyecto. Este código se ejecutará cuando se inicie la aplicación de consola.

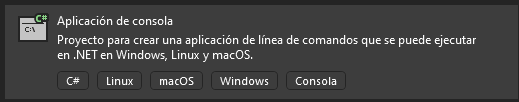


Figura 5. Aplicación de consola C#.

Las aplicaciones de consola son útiles para tareas simples y automatizadas que no requieren una interfaz gráfica de usuario. Por ejemplo, se puede crear una aplicación de consola que convierta un archivo de texto a mayúsculas o que calcule el promedio de un conjunto de números, entre otros.

## APLICACIÓN DE WPF (.NET FRAMEWORK)

WPF (Windows Presentation Foundation) es un marco de desarrollo de aplicaciones de interfaz de usuario para Windows que se utiliza para crear aplicaciones gráficas. Es parte del .NET Framework de Microsoft y se puede utilizar con lenguajes de programación como C# y Visual Basic.

Una aplicación de WPF es un tipo de aplicación de Windows que utiliza la tecnología WPF para crear una interfaz de usuario gráfica. Esto significa que la aplicación se puede mostrar en una ventana y contiene elementos gráficos como botones, menús y gráficos. WPF proporciona una gran cantidad de características y herramientas para crear aplicaciones atractivas y modernas con una experiencia de usuario fluida.

Para crear una aplicación de WPF, es necesario instalar el .NET Framework y utilizar un entorno de desarrollo como Visual Studio. Luego, se deberá crear un nuevo proyecto de aplicación de WPF y finalmente se podrá escribir en código C# o Visual Basic en el archivo principal del proyecto. También es posible utilizar el diseñador de interfaz de usuario de Visual Studio para arrastrar y colocar elementos gráficos en la aplicación.

Interfaz de usuario gráfica, Texto, Aplicación, Sitio web

Descripción generada automáticamente

Figura 6. Aplicación de WPF (.NET Framework)

## ARCHIVOS EN WPF (.NET FRAMEWORK)

### **XAMARIN FORMS**

XAML es un lenguaje de marcado basado en XML desarrollado por Microsoft como una forma de crear objetos, inicializarlos y organizarlos en jerarquías de padres e hijos sin necesidad de utilizar código imperativo. Se ha adaptado a diversas tecnologías del .NET Framework, como Windows Presentation Foundation (WPF), Silverlight, Windows Runtime, la Plataforma Universal de Windows (UWP) y Xamarin.Forms. En el archivo XAML, los desarrolladores de Xamarin.Forms pueden definir interfaces de usuario utilizando controles (vistas), diseños (maquetaciones), páginas o clases personalizadas.

El archivo XAML puede ser compilado o incrustado en el archivo ejecutable de la aplicación. En cualquier caso, la información XAML se analiza en tiempo de compilación para localizar objetos que hayan sido nombrados y nuevamente en tiempo de ejecución para instanciar e inicializar objetos y para establecer enlaces entre ellos y el código de programación. Esto permite que los desarrolladores creen interfaces de usuario de forma sencilla y rápida utilizando un lenguaje de marcado en lugar de tener que escribir código imperativo. Además, al utilizar XAML, es posible separar la lógica del programa de la interfaz de usuario, lo que facilita el mantenimiento y la actualización de la aplicación.

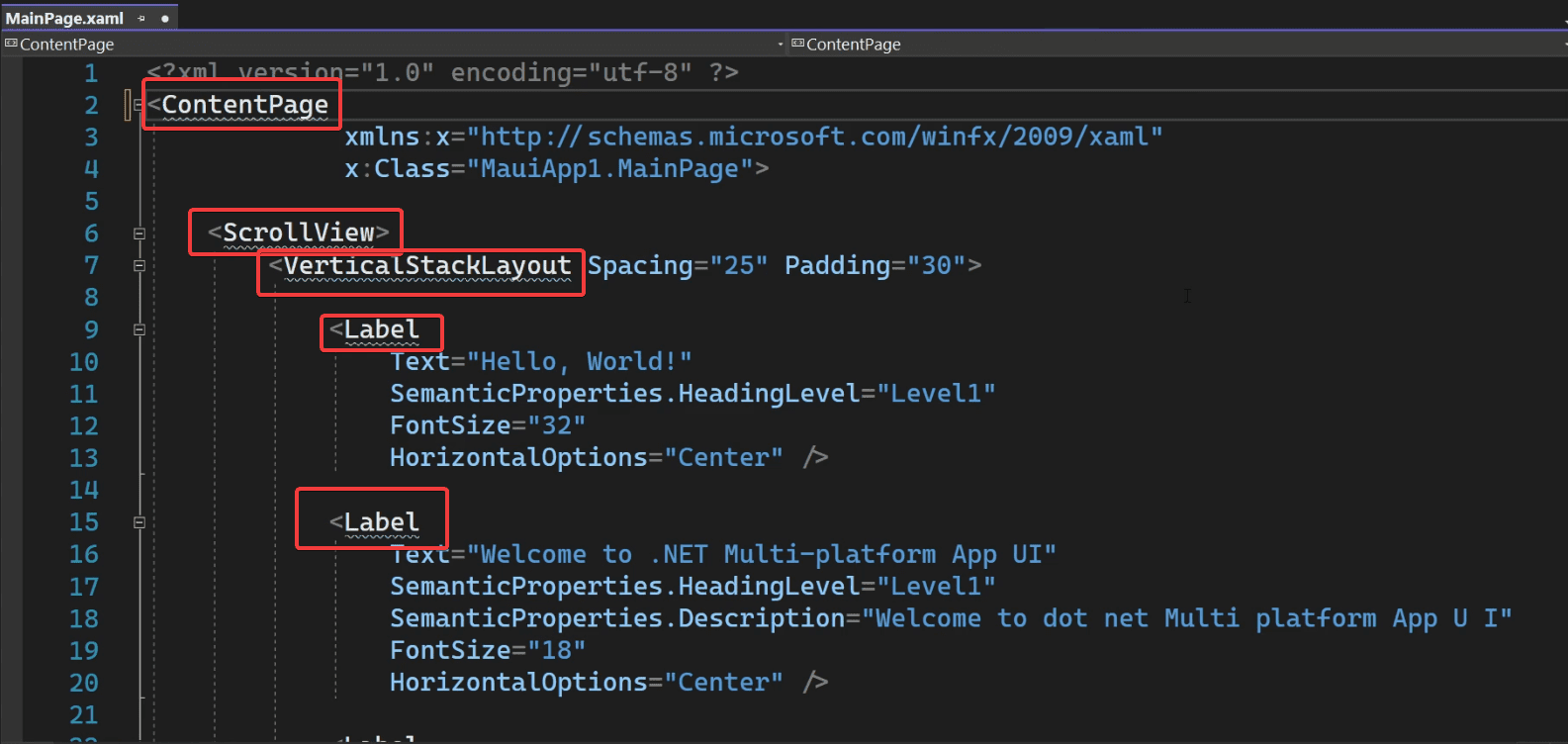


Figura 7. Ejemplo de un archivo XAML.

#### **ANATOMÍA DE UN ARCHIVO XAML**

* App.xaml: es el archivo principal, de marcado.
* App.xaml.cs: es el archivo de código subyacente de C# asociado al archivo XAML.

Los dos archivos contribuyen a una clase denominada App que se deriva de Application.

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

Figura 8. Anatomía de un archivo xaml.

#### **SINTAXÍS DE XAML**

XAML es un lenguaje de marcas utilizado para crear interfaces de usuario en aplicaciones de Windows. Se utiliza principalmente para crear e inicializar objetos en una aplicación, pero también tiene características avanzadas que permiten establecer propiedades en objetos complejos.

La sintaxis de XAML tiene dos características esenciales para establecer propiedades en objetos complejos: los elementos de propiedad y las propiedades adjuntas. Los elementos de propiedad permiten establecer valores para propiedades de un objeto que no se pueden representar fácilmente como cadenas XML. Por ejemplo, si queremos establecer un color de fondo para un botón, podemos utilizar el elemento de propiedad "Background" y establecer su valor en una cadena de color hexadecimal.

Por otro lado, las propiedades adjuntas permiten establecer propiedades en una clase secundaria que no están definidas en la clase principal. Por ejemplo, si queremos establecer una propiedad "IsSelected" en un elemento de lista, podemos utilizar una propiedad adjunta para hacerlo.

#### **VENTAJAS DE XAML**

* El código XAML suele ser más compacto y legible que el código equivalente.
* La jerarquía inherente Padre e Hijo de XML permite a XAML imitar con mayor claridad visual la jerarquía Padre e Hijo de los objetos de la interfaz de usuario.
* El código XAML puede ser fácilmente escrito de forma manual por los programadores, pero también puede ser generado por herramientas visuales como Microsoft Blend.

#### **DESVENTAJAS DE XAML**

* XAML no puede contener código imperativo. Todos los controladores de eventos deben ser definidos en un archivo de código.
* XAML no puede contener ciclos para procesamiento repetitivo. Sin embargo, varios objetos visuales de Xamarin.Forms pueden generar múltiples elementos como por ejemplo el objeto ListView que puede generar varios elementos dependiendo de los objetos en su colección ItemsSource.
* XAML no puede contener procesamiento condicional, sin embargo, un enlace de datos puede hacer referencia a un convertidor de enlace basado en código que permite un procesamiento condicional de forma eficaz.
* Generalmente, XAML no puede crear instancias de clases que no definen un constructor sin parámetros. Sin embargo, algunas veces existe un mecanismo para evitar esta restricción.
* Generalmente, XAML no puede invocar métodos. Sin embargo, algunas veces también existe un mecanismo para evitar esta restricción.
* Aún no existe un diseñador visual para generar código XAML en aplicaciones Xamarin.Forms.

#### **GUÍA RÁPIDA DE CONTROLES XAML**

Tabla 1. Vista para presentación.

|  |  |  |  |
| --- | --- | --- | --- |
| # | Nombre | Definición | Etiqueta |
| 1 | BoxView | Muestra un rectángulo de un color determinado. | <BoxView Color="Accent"      WidthRequest="150"      HeightRequest="150"      HorizontalOptions="Center"> |
| 2 | Ellipse | Muestra una elipse o un círculo. | <Elipse Fill="Red"      WidthRequest="150"      HeightRequest="50"      HorizontalOptions="Center" /> |
| 3 | Image | Muestra un mapa de bits. | <Image Source="https://aka.ms/campus.jpg"      Aspect="AspectFit"      HorizontalOptions="Center" /> |
| 4 | Etiqueta | Muestra una o varias líneas de texto. | <Label Text="Hello, Xamarin.Forms!"      FontSize="Large"      FontAttributes="Cursiva"      HorizontalTextAlignment="Center" /> |
| 5 | Línea | Mostrar una línea. | <Línea X1="40"      Y1="0"      X2="0"      Y2="120"      Stroke="Red"      HorizontalOptions="Center" /> |
| 6 | Map | Muestra un mapa. | <maps:Map ItemsSource="{Binding Locations}" /> |
| 7 | Ruta de acceso | Muestras curvas y formas complejas. | <Path Stroke="Black"      Aspect="Uniform"      HorizontalOptions="Center"      HeightRequest="100"      WidthRequest="100"      Data="M13.9,16.2              L32,16.2 32,31.9 13.9,30.1Z              M0,16.2              L11.9,16.2 11.9,29.9 0,28.6Z              M11.9,2              L11.9,14.2 0,14.2 0,3.3Z              M32,0              L32,14.2 13.9,14.2 13.9,1.8Z" /> |
| 8 | Polygon | Muestra un polígono | <Polygon      Points="0 48, 0 144, 96 150, 100 0, 192 0, 192 96,                   50 96, 48 192, 150 200 144 48"      Fill="Blue"      Stroke="Red"      StrokeThickness="3"      HorizontalOptions="Center" /> |
| 9 | Polilínea | Muestra una serie de líneas rectas conectadas. | <Puntos polilínea="0,0 10,30, 15,0 18,60 23,30 35,30 40,0                    43,60 48,30 100,30"      Stroke="Red"      HorizontalOptions="Center" /> |
| 10 | Rectángulo | Muestra un rectángulo o cuadrado. | <Rectangle Fill="Red"      WidthRequest="150"      HeightRequest="50"      HorizontalOptions="Center" /> |
| 11 | WebView | Muestra páginas web o contenido HTML. | <WebView Source="https://learn.microsoft.com/xamarin/"      VerticalOptions="FillAndExpand" /> |

Tabla 2. Vista para que inicien comandos.

|  |  |  |  |
| --- | --- | --- | --- |
| # | Nombre | Definición | Etiqueta |
| 1 | Botón | Muestra texto en un objeto rectangular. | <Button Text="Click Me!"      Font="Large"      BorderWidth="1"      HorizontalOptions="Center"      VerticalOptions="CenterAndExpand"      Clicked="OnButtonClicked" /> |
| 2 | ImageButton | Muestra una imagen en un objeto rectangular. | <ImageButton Source="XamarinLogo.png"      HorizontalOptions="Center"      VerticalOptions="CenterAndExpand"      Clicked="OnImageButtonClicked" /> |
| 3 | RadioButton | Permite la selección de una opción de un conjunto. | <RadioButton Text="Piña"      CheckedChanged="OnRadioButtonCheckedChanged" /> |
| 4 | RefreshView | Proporciona funcionalidad de extracción para actualizar contenido desplazable. | <RefreshView IsRefreshing="{Binding IsRefreshing}"      Command="{Binding RefreshCommand}">      <!-- control desplazable aquí:>  </RefreshView> |
| 5 | Barra de búsqueda | Acepta la entrada del usuario que usa para realizar una búsqueda. | <SearchBar Placeholder="Enter search term"      SearchButtonPressed="OnSearchBarButtonPressed" /> |
| 6 | SwipeView | Proporciona elementos de menú contextual que se muestran mediante un gesto de deslizar el dedo. | <SwipeView>      <SwipeView.LeftItems>          <SwipeItems>              <SwipeItem Text="Delete"                  IconImageSource="delete.png"                  BackgroundColor="LightPink"                  Invoked="OnDeleteInvoked" />          </SwipeItems>      </SwipeView.LeftItems>      <!--Contenido-->  </SwipeView> |

Tabla 3. Vista para establecer valores.

|  |  |  |  |
| --- | --- | --- | --- |
| # | Nombre | Definición | Etiqueta |
| 1 | CheckBox | Permite la selección de un boolean valor. | <CheckBox IsChecked="true"      HorizontalOptions="Center"      VerticalOptions="CenterAndExpand" /> |
| 2 | Control deslizante | Permite la selección de un double valor de un intervalo continuo. | <Slider Minimum="0"      Maximum="100"      VerticalOptions="CenterAndExpand" /> |
| 3 | Control de incremento | Permite la selección de un double valor de un intervalo incremental. | <Stepper Minimum="0"      Maximum="10"      Increment="0.1"      HorizontalOptions="Center"      VerticalOptions="CenterAndExpand" /> |
| 4 | Switch | Permite la selección de un boolean valor. | <Switch IsToggled="false"      HorizontalOptions="Center"      VerticalOptions="CenterAndExpand" /> |
| 5 | DatePicker | Permite la selección de una fecha. | <DatePicker Format="D"      VerticalOptions="CenterAndExpand" /> |
| 6 | TimePicker | Permite la selección de una hora. | <TimePicker Format="T"      VerticalOptions="CenterAndExpand" /> |

Tabla 4. Vistas para editar texto

|  |  |  |  |
| --- | --- | --- | --- |
| # | Nombre | Definición | Etiqueta |
| 1 | Entrada | Permite escribir y editar una sola línea de texto. | <Entry Keyboard="Email"      Placeholder="Escriba la dirección de correo electrónico"      VerticalOptions="CenterAndExpand" /> |
| 2 | Editor | Permite escribir y editar varias líneas de texto. | <Editor VerticalOptions="FillAndExpand" /> |

Tabla 5. Vista para indicar actividad

|  |  |  |  |
| --- | --- | --- | --- |
| # | Nombre | Definición | Etiqueta |
| 1 | ActivityIndicator | Muestra una animación para mostrar que la aplicación se dedica a una actividad larga, sin proporcionar ninguna indicación de progreso. | <ActivityIndicator IsRunning="True"      VerticalOptions="CenterAndExpand" /> |
| 2 | ProgressBar | Muestra una animación para mostrar que la aplicación progresa a través de una actividad larga. | <ProgressBar Progress=".5"      VerticalOptions="CenterAndExpand" /> |

Tabla 6. Vistas para mostrar colecciones.

|  |  |  |  |
| --- | --- | --- | --- |
| # | Nombre | Definición | Etiqueta |
| 1 | CarouselView | Muestra una lista desplazable de elementos de datos. | <CarouselView ItemsSource="{Binding Monkeys}">  ItemTemplate="{StaticResource MonkeyTemplate}" /> |
| 2 | CollectionView | Muestra una lista desplazable de elementos de datos seleccionables mediante diferentes especificaciones de diseño. | <CollectionView ItemsSource="{Binding Monkeys}">  ItemTemplate="{StaticResource MonkeyTemplate}"  ItemsLayout="VerticalGrid, 2" /> |
| 3 | IndicatorView | Muestra indicadores que representan el número de elementos de un CarouselView objeto. | <IndicatorView x:Name="indicatorView"      IndicatorColor="LightGray"      SelectedIndicatorColor="DarkGray" /> |
| 4 | ListView | Muestra una lista desplazable de elementos de datos seleccionables. | <ListView ItemsSource="{Binding Monkeys}">  ItemTemplate="{StaticResource MonkeyTemplate}" /> |
| 5 | Selector | Muestra un elemento de selección de una lista de cadenas de texto. | <Picker Title="Select a monkey"      TitleColor="Red">      <Picker.ItemsSource>          <x:Array Type="{x:Type x:String}">              <x:String>Baboon</x:String>              <x:String>Capuchin Monkey</x:String>              <x:String>Blue Monkey</x:String>              <x:String>Squirrel Monkey</x:String>              <x:String>Golden Lion Tamarin</x:String>              <x:String>Howler Monkey</x:String>              <x:String Japanese>Macaque</x:String>          </x:Array>      </Picker.ItemsSource>  </Picker> |
| 6 | TableView | Muestra una lista de filas interactivas. | <TableView Intent="Settings">      <TableRoot>          <TableSection Title="Ring">              <SwitchCell Text="New Voice Mail" />              <SwitchCell Text="New Mail" On="true" />          </TableSection>      </TableRoot>  </TableView> |

### **ARCHIVO DE RECURSOS .RESX**

Los archivos de recursos en formato XML (.resx) deben cumplir con ciertas reglas de formato y estructura, lo que puede dificultar su creación manual. Una opción es utilizar la biblioteca de clases .NET Framework para crear estos archivos y acceder a los recursos que contienen.

Donde un archivo de recursos .resx funciona como una base de datos de diversos tipos de recursos los cuales pueden ser:

* Cadenas
* Imágenes
* Iconos
* Audios
* Archivos

Un archivo .resx es un tipo de archivo que se utiliza para almacenar recursos, como cadenas de texto y pares clave-valor, que pueden ser utilizados por una aplicación o componente. Estos archivos son utilizados a menudo para almacenar texto que se mostrará al usuario, como mensajes de error o cadenas de interfaz de usuario, de manera que pueden ser fácilmente localizados y modificados.

#### **CARACTERÍSTICAS DE UN ARCHIVO RESX**

* Formato de texto plano: los archivos .resx se almacenan en formato de texto plano, lo que significa que pueden ser leídos y editados con cualquier editor de texto.
* Estructura de pares clave-valor: los recursos se almacenan en el archivo .resx como pares clave-valor, donde la clave es un nombre único que se utiliza para identificar el recurso y el valor es el contenido del recurso en sí.
* Compilación: los archivos .resx se compilan en archivos .resources, que son archivos binarios que contienen los recursos de la aplicación. Estos archivos .resources se incluyen en el ensamblado de la aplicación y se pueden utilizar para recuperar los recursos en tiempo de ejecución.
* Localización: los archivos .resx se pueden utilizar para facilitar la localización de una aplicación, lo que significa que se pueden crear versiones de los recursos en diferentes idiomas y utilizarlos según el idioma seleccionado por el usuario.



Figura 9. Archivos de Recursos

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 10. Tipos de archivos

## SERIALIZACIÓN Y DESERIALIZACIÓN

* Serialización: es el proceso de convertir un objeto en secuencia de bytes que puede ser almacenado en un archivo o transmitido por medio de la red.
* Deserialización: es el proceso inverso a la serialización se encarga de convertir una secuencia de bytes en un objeto.

Para hacer uso de la serialización y deserialización en C# se puede hacer uso de las clases: BinaryFormatter, XmlSerializer, DataContractSerializer, entre otros.

En este proyecto se hará uso de la clase BinaryFormatter.

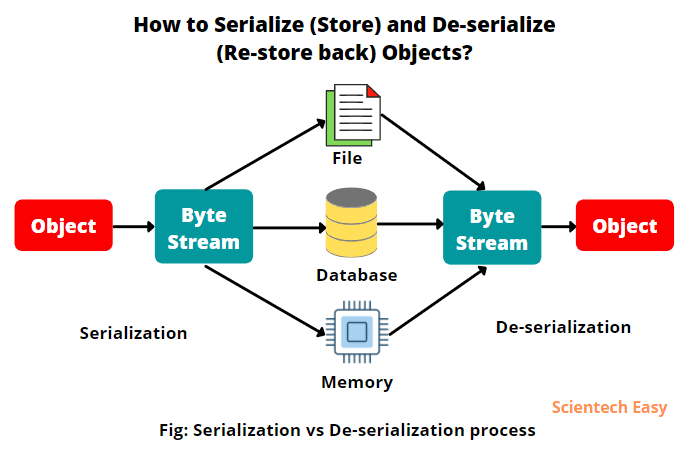


Figura 11. Serialización y deserialización.

### **SERIALIZACIÓN Y DESERIALIZACIÓN CON BINARYFORMATTER**

La clase BinaryFormatter en C# es una herramienta importante para la serialización y deserialización de objetos. La serialización es el proceso de convertir un objeto en un formato binario, que puede ser almacenado en un archivo o transmitido a través de una red. La deserialización, por otro lado, es el proceso de convertir un formato binario en un objeto.

La clase BinaryFormatter proporciona métodos para realizar estas operaciones, tales como Serialize y Deserialize. Con estos métodos, se pueden convertir objetos en un formato binario y viceversa. Esto es especialmente útil cuando se trabaja con aplicaciones distribuidas o en las que se requiere almacenar o transmitir objetos.

BinaryFormatter es una clase que se encuentra en el espacio de nombres System.Runtime.Serialization.Formatter.Binary, parte de la biblioteca base de C#. Esto significa que no se requiere ninguna configuración adicional para utilizarlo en un proyecto, ya que ya está disponible en el lenguaje.

## ESPACIO DE NOMBRES

Los espacios de nombres en C# son una forma de organizar y agrupar tipos (Clases, estructuras, enumeraciones, entre otros) de forma lógica, donde cada espacio de nombres puede contener un conjunto de tipos relacionados.Los espacios de nombres permiten evitar conflictos de nombres entre tipos diferentes bibliotecas o proyectos y facilitan la referencia a los tipos utilizando un nombre completo.

Por ejemplo: el tipo “System.IO.File” pertenece al espacio de nombres “System.IO” y se refiere al tipo “File” dentro de este espacio de nombres.

Un programa de C# puede utilizar varios espacios de nombres, ya sea mediante la declaración "using" en la parte superior del archivo o mediante la referencia al nombre completo del tipo cada vez que se utiliza.

Los espacios de nombres también se pueden anidar, lo que permite una organización jerárquica de los tipos. Por ejemplo, el espacio de nombres "System.IO" contiene varios tipos relacionados con la entrada/salida, como "File" y "Directory".

# PARTE PRÁCTICA

## 4.1 CREACIÓN DE CARPETAS PARA EL PROYECTO

Para comenzar con el desarrollo del proyecto es importante crear las carpetas que nos permitan tener el proyecto organizado de tal forma que se separe la aplicación cliente, de la aplicación servidor y la documentación del mismo, para ello se tendrá la carpeta principal que es SOCKETS\_C#\_GRUPO5 y dentro de esta se tendrá dos subcarpetas que son APLICATIVO y DOCUMENTACIÓN, y dentro de APLICATIVO se tendrá dos subcarpetas que serán CLIENTE y SERVIDOR.

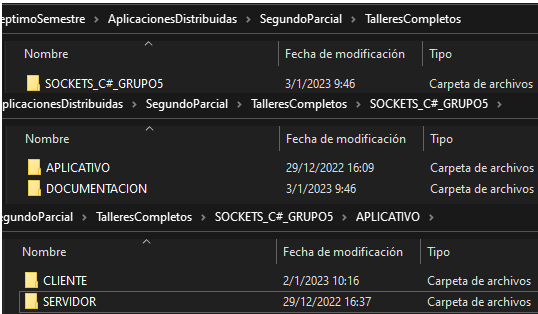


Figura 12. Estructura del proyecto.

## PROYECTO SERVIDOR

### **CREACIÓN DEL PROYECTO SERVIDOR**

Para comenzar con el desarrollo del proyecto Servidor se deberá abrir Visual Studio 2022 y hacer clic en Crear un proyecto.

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

Figura 13. Pantalla principal de Visual Studio 2022.

Una vez seleccionado el Crear un proyecto se mostrará la siguiente pantalla donde se deberá realizar la búsqueda de un proyecto Aplicación de consola, y dar clic en siguiente.

Texto

Descripción generada automáticamente

Figura 14. Selección del tipo de proyecto.

Finalmente, se deberá configurar los parámetros del proyecto como es el Nombre del proyecto, la ubicación que será en la carpeta APLICATIVO/SERVIDOR que fueron creadas anteriormente, el nombre de la solución y el Marco (Versión de .NET Framework que se va a utilizar en el proyecto), finalmente se hace clic en crear.

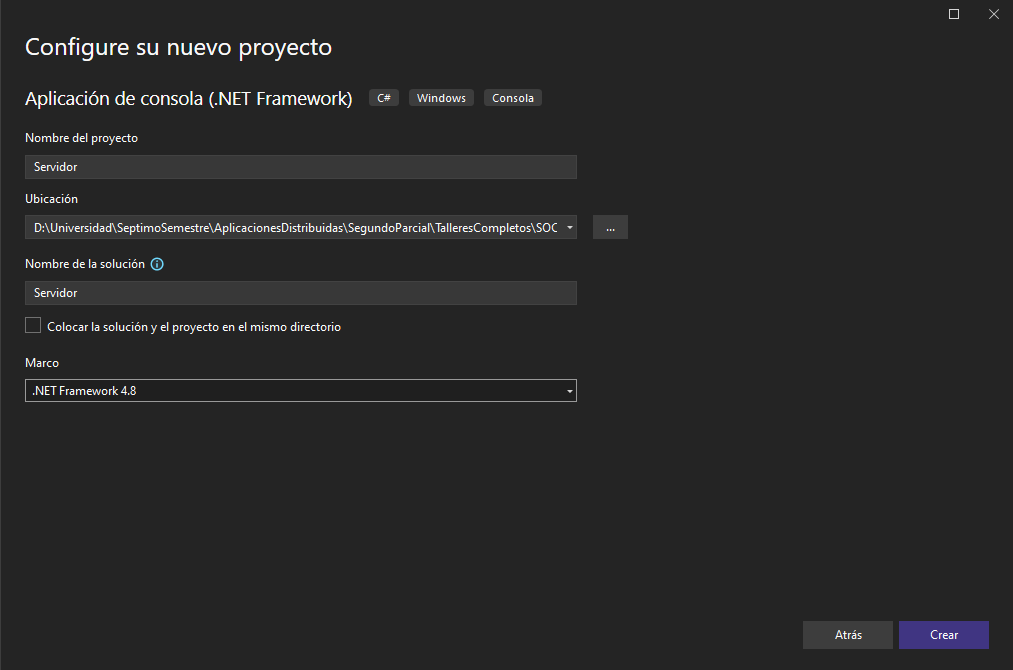


Figura 15. Configuración del proyecto Servidor.

Una vez creado el proyecto tenemos la siguiente estructura, donde Program.cs es la clase principal.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Figura 16. Estructura de un proyecto Aplicación de consola.

### **ESTRUCTURA DEL PROYECTO SERVIDOR (MODELO-VISTA-CONTROLADOR)**

Para continuar con el desarrollo se deberá reestructurar el proyecto creado con la finalidad de aplicar el Patrón MODELO-VISTA-CONTROLADOR que nos permiten tener el proyecto organizado y de esta forma sea entendible y escalable.

Donde se procede hacer clic derecho sobre el proyecto, seguido de eso se selecciona la opción de Agregar, y finalmente Nueva Carpeta como se muestra en la siguiente ilustración.

Texto

Descripción generada automáticamente

Figura 17. Creación de carpetas Modelo, Vista, Controlador

Además, de las carpetas Modelo, Vista, Controlador se creará una carpeta adicional que será Utils, la cual se encarga de almacenar archivos que nos permiten realizar acciones en específico durante el desarrollo del proyecto.

Interfaz de usuario gráfica, Texto, Chat o mensaje de texto

Descripción generada automáticamente

Figura 18. Estructura del proyecto.

### **CREACIÓN DE ARCHIVOS PARA CADA CARPETA**

Para continuar con el desarrollo del proyecto se crean los archivos pertenecientes a cada carpeta del proyecto como se muestra a continuación.

#### **MODELO**

La carpeta modelo contiene dos archivos los cuales son Message y User donde para crear los archivos se deberá hacer clic derecho sobre la carpeta Modelo, seguido de eso en agregar y elemento nuevo como se muestra en la siguiente ilustración.

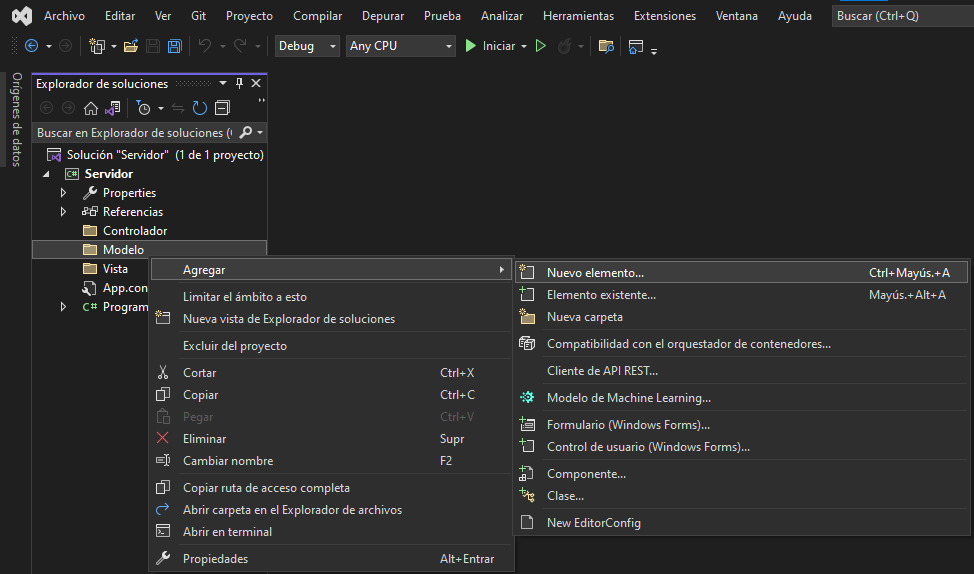


Figura 19. Creación de archivos en la carpeta Modelo.

Se procede a seleccionar el tipo de archivo que se desea agregar y se le asigna el nombre al archivo como se muestra en la siguiente ilustración.

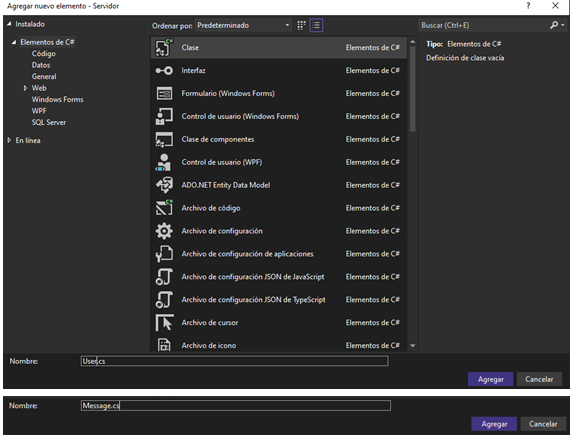


Figura 20. Selección de tipo de archivo y nombre.

#### **VISTA**

La carpeta Vista contiene un archivo que será el que muestre el primer mensaje por parte del servidor al usuario, para ello se procede a realizar el mismo proceso para la creación de archivo en la carpeta Modelo, se hace clic derecho sobre la carpeta Vista, agregar y elemento nuevo.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 21. Creación de archivos en la carpeta Vista.

Se procede a seleccionar el tipo de archivo que se desea agregar y se le asigna el nombre al archivo como se muestra en la siguiente ilustración.

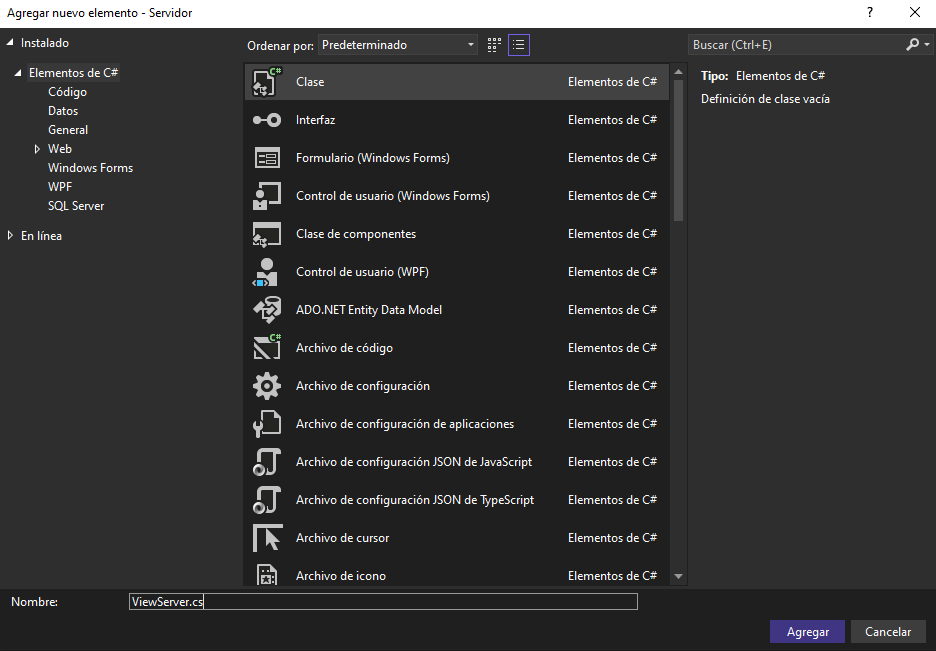


Figura 22. Selección de tipo de archivo y nombre.

#### **CONTROLADOR**

Para crear los archivos en la carpeta Controlador, se hace clic derecho en la Carpeta controlador, agregar y elemento nuevo.

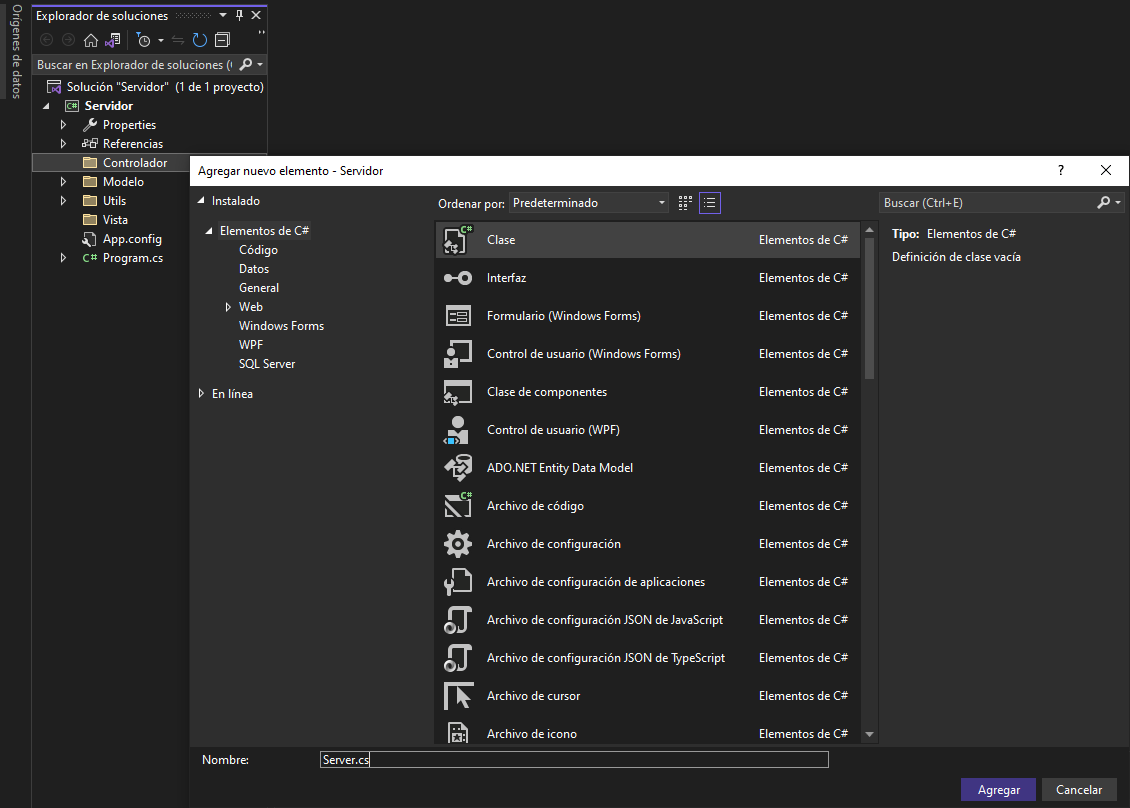


Figura 23. Selección del tipo de archivo y asignación de nombre.

#### **UTILS**

Finalmente se crea los archivos de la carpeta Utils, se hace clic derecho sobre la carpeta, agregar y elemento nuevo.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 24. Selección del tipo de archivo y asignación de nombre.

Una vez terminado la creación de los archivos que son parte del proyecto Servidor se tiene la siguiente estructura.

Texto, Chat o mensaje de texto

Descripción generada automáticamente

Figura 25. Estructura del proyecto Servidor, Carpetas y Archivos.

### **CODIFICACIÓN DEL PROYECTO SERVIDOR**

#### **MODELO – USER**

Tabla 7. Codificación del archivo User.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Chat

{

[Serializable]

public class User

{

public string id;

public string nick;

public User(string id, string nick)

{

this.id = id;

this.nick = nick;

}

}

}

#### **MODELO – MESSAGE**

Tabla 8. Codificación del archivo Message.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Chat

{

[Serializable]

public class Message

{

public User from;

public User to;

public string msg;

public Message(User from, User to, string msg)

{

this.from = from;

this.to = to;

this.msg = msg;

}

}

}

#### **VISTA – VIEWSERVER**

Tabla 9. Codificación del archivo ViewServer.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Servidor.Vista

{

public class ViewServer

{

private string message { get; set; }

public ViewServer()

{

this.message = message;

}

public void showMessage()

{

this.message = "Servidor esperando usuarios.....";

Console.WriteLine(message);

}

}

}

#### **CONTROLADOR – SERVER**

Tabla 10. Codificación del archivo Server.cs

using System;

using System.Collections;

using System.Net.Sockets;

using System.Net;

using System.Threading;

using Serialization;

using Chat;

namespace Servidor.Controlador

{

class Server

{

Socket socket;

Thread listenThread;

Hashtable usersTable;

public Server()

{

try

{

IPHostEntry host = Dns.GetHostEntry("localhost");

IPAddress addr = host.AddressList[0];

IPEndPoint endPoint = new IPEndPoint(addr, 4404);

socket = new Socket(endPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

socket.Bind(endPoint);

socket.Listen(10);

listenThread = new Thread(this.Listen);

listenThread.Start();

usersTable = new Hashtable();

}

catch (Exception e)

{

Console.WriteLine("{0}", e.Message);

}

}

private void Listen()

{

Socket client;

while (true)

{

client = this.socket.Accept();

listenThread = new Thread(this.ListenClient);

listenThread.Start(client);

}

}

private void ListenClient(object o)

{

Socket client = (Socket)o;

object received;

do

{

received = this.Receive(client);

Console.WriteLine(((Chat.User)received).nick + " en línea...");

} while (!(received is User));

this.usersTable.Add(received, client);

this.BroadCast(received);

this.SendAllUsers(client);

while (true)

{

received = this.Receive(client);

if (received is Message)

{

if (!System.IO.Path.IsPathRooted(((Message)received).msg))

{

Console.WriteLine("De: " + ((Message)received).@from.nick +

" \nPara: " + ((Message)received).to.nick +

" \nMensaje: " + ((Message)received).msg);

this.SendMessage((Message)received);

}

}

}

}

private void BroadCast(object o)

{

foreach (DictionaryEntry d in this.usersTable)

{

this.Send((Socket)d.Value, o);

}

}

private void SendAllUsers(Socket s)

{

foreach (DictionaryEntry d in this.usersTable)

{

this.Send(s, d.Key);

}

}

private void SendMessage(Message m)

{

User tmpUser;

foreach (DictionaryEntry d in this.usersTable)

{

tmpUser = (User)d.Key;

if (tmpUser.id == m.to.id)

{

this.Send((Socket)d.Value, m);

break;

}

}

}

private void Send(Socket s, object o)

{

byte[] buffer = new byte[1024];

byte[] obj = BinarySerialization.Serializate(o);

Array.Copy(obj, buffer, obj.Length);

s.Send(buffer);

}

private object Receive(Socket s)

{

byte[] buffer = new byte[1024];

s.Receive(buffer);

return BinarySerialization.Deserializate(buffer);

}

}

}

#### **UTILS – SERIALIZATION**

Tabla 11. Codificación del archivo Serialization.cs

using System;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using System.Runtime.Serialization;

using System.Reflection;

namespace Serialization

{

class BinarySerialization

{

public static byte[] Serializate(object toSerializate)

{

MemoryStream memory = new MemoryStream();

BinaryFormatter formatter = new BinaryFormatter();

formatter.Serialize(memory, toSerializate);

return memory.ToArray();

}

public static object Deserializate(byte[] data)

{

MemoryStream memory = new MemoryStream(data);

BinaryFormatter formatter = new BinaryFormatter();

formatter.Binder = new CurrentAssemblyDeserializationBinder();

return formatter.Deserialize(memory);

}

}

public class CurrentAssemblyDeserializationBinder : SerializationBinder

{

public override Type BindToType(string assemblyName, string typeName)

{

return Type.GetType(String.Format("{0}, {1}", typeName, Assembly.GetExecutingAssembly().FullName));

}

}

}

Finalmente se realiza la codificación del archivo Program.cs que es la clase principal del proyecto.

Tabla 12. Codificación del archivo Program.cs

using Servidor.Controlador;

using Servidor.Vista;

using System;

namespace Servidor

{

internal class Program

{

static void Main(string[] args)

{

Server serverserver = new Server();

ViewServer servidor = new ViewServer();

servidor.showMessage();

Console.ReadKey();

}

}

}

## PROYECTO CLIENTE

### **CREACIÓN DEL PROYECTO CLIENTE**

Para comenzar con el desarrollo del proyecto Cliente se debe de considerar que se va a utilizar un tipo de aplicación diferente a la del Servidor ya que el proyecto Cliente requiere de una interfaz gráfica de usuario que los permita interactuar con otros usuarios por medio de mensajes, y envío de archivos de texto, imágenes y audio.

Se deberá abrir Visual Studio 2022 y seleccionar en Crear un proyecto como se muestra en la siguiente ilustración.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 26. Pantalla principal de Visual Studio 2022.

Una vez seleccionada la opción Crear un proyecto se muestra la siguiente ilustración donde se deberá buscar la aplicación de tipo “Aplicación de WPF (.NET Framework)” la cual nos permite hacer uso de XAML para interfaces de usuario.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 27. Selección del tipo de aplicación para el proyecto Cliente.

Una vez seleccionado el tipo de aplicación que se va a crear para el proyecto Cliente, se procede a configurar los parámetros del Proyecto los cuales son, nombre del proyecto, la ubicación que es en la Carpeta APLICATIVO/CLIENTE, se le asigna un nombre a la solución y se selecciona el Marco (Versión de .NET que se va a utilizar).

Captura de pantalla de un celular

Descripción generada automáticamente

Figura 28. Configuración de parámetros del proyecto Cliente.

Finalmente se tiene la siguiente estructura del proyecto Cliente.

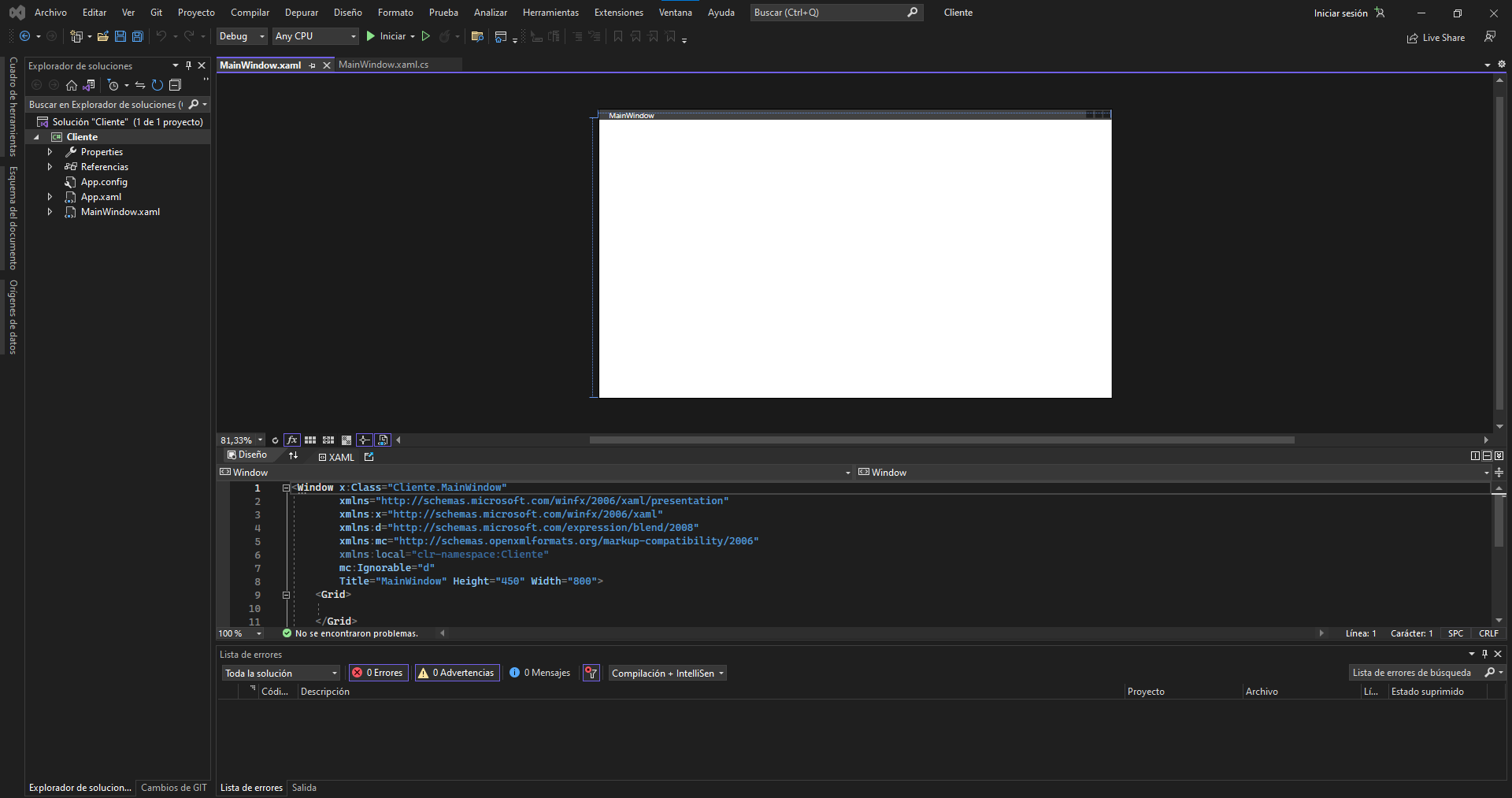


Figura 29. Estructura del proyecto Cliente.

### **ESTRUCTURA DEL PROYECTO CLIENTE (MODELO-VISTA-CONTROLADOR)**

Para comenzar con el desarrollo del proyecto Cliente se deberá realizar una reestructuración creando las carpetas principales que son MODELO, VISTA y CONTROLADOR las cuales nos permitirán tener el proyecto organizado y con esto sea entendible y escalable.

Para ellos se procede hacer clic derecho sobre el proyecto, seleccionar agregar y Nueva Carpeta como se muestra en la siguiente ilustración.

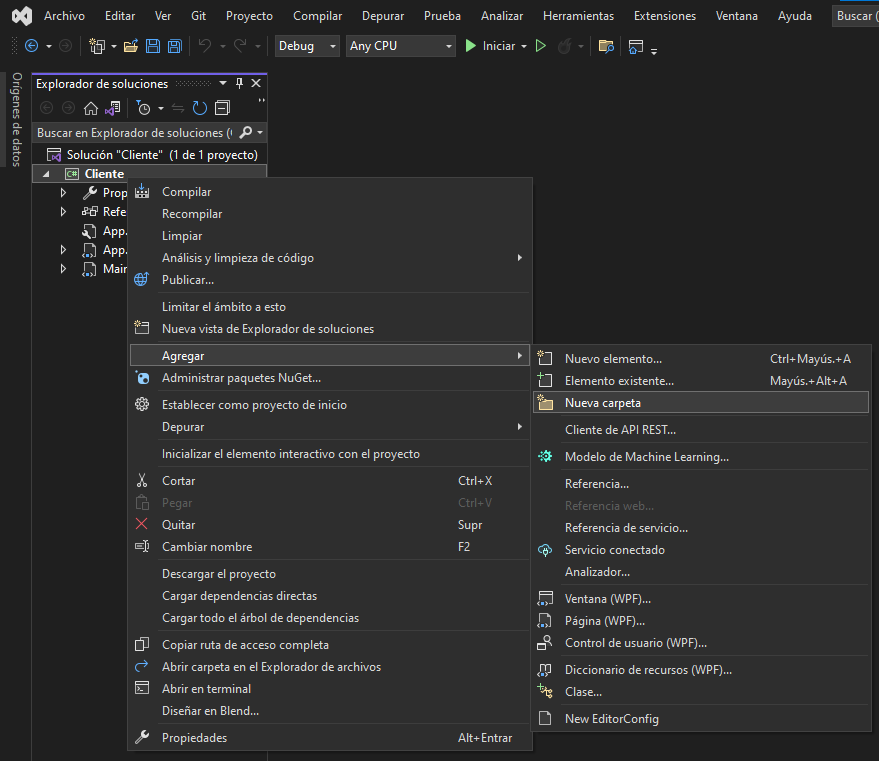


Figura 30. Creación de carpetas Modelo, Vista y Controlador.

Además de las carpetas Modelo, Vista y Controlador se crearán otras carpetas que nos permiten organizar de mejor manera el proyecto las cuales serán, Resources, Utils, Fonts, Imágenes, Components donde cada una cumple una función en específico dentro del proyecto.

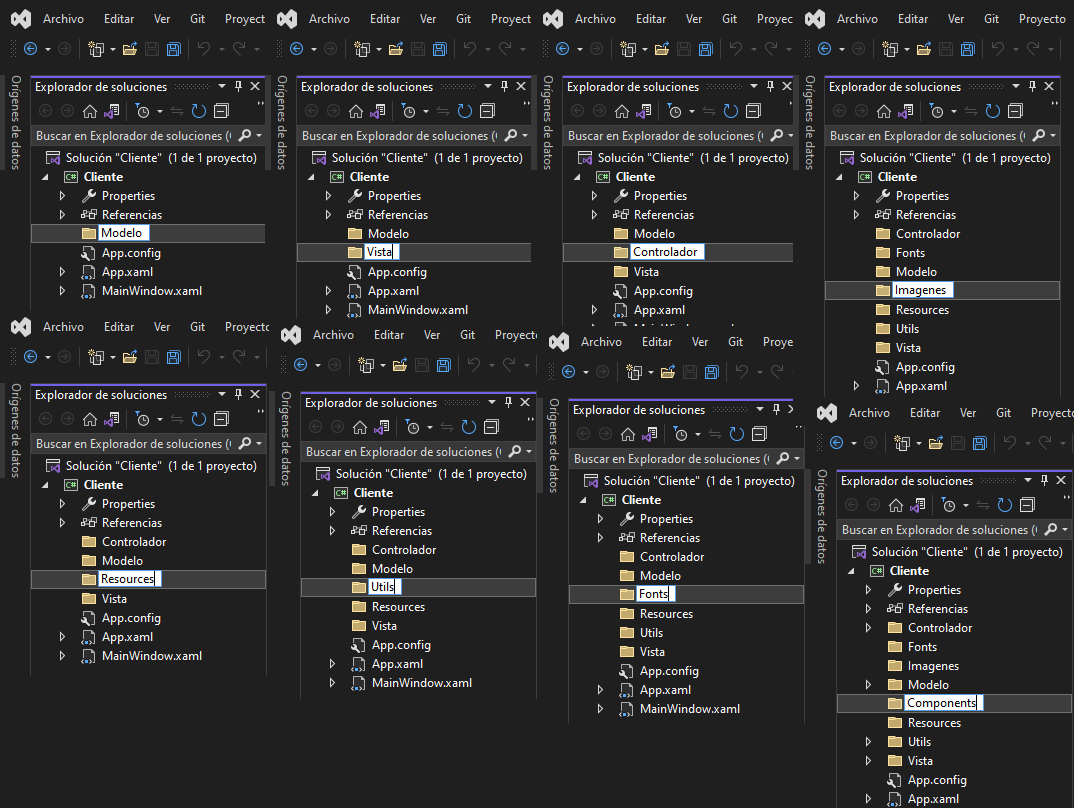


Figura 31. Creación de carpetas para el proyecto Cliente.

Una vez creadas las distintas carpetas se tiene la siguiente estructura de proyecto.

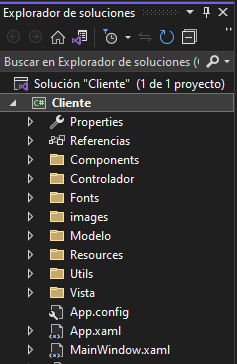


Figura 32. Estructura del proyecto Cliente.

### **CREACIÓN DE ARCHIVOS PARA CADA CARPETA**

#### **MODELO**

La carpeta modelo contiene dos archivos principales los cuales son Message y User, y para poder crearlos se procede hacer clic derecho sobre la carpeta Modelo, agregar y elemento nuevo como se muestra en la siguiente ilustración.

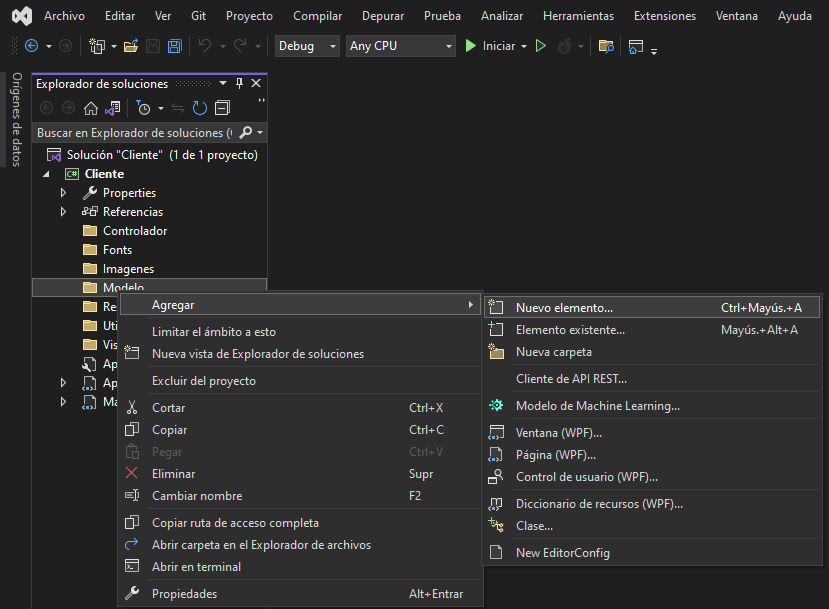


Figura 33. Agregar nuevo elemento a la carpeta Modelo.

Una vez seleccionado la opción Nuevo Elemento se mostrará la siguiente pantalla donde se selecciona el tipo de archivo que se va a crear y se le asigna el nombre.

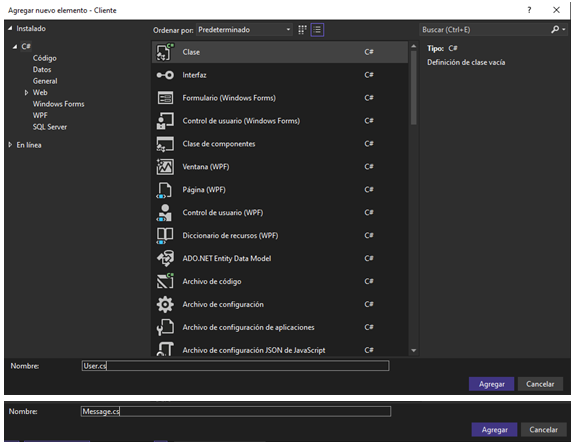


Figura 34. Selección del tipo de archivo y asignación de nombre

#### **VISTA**

Para poder agregar un archivo a la carpeta Vista se realiza un proceso similar al que se realizó en la carpeta Modelo, para ello se hace clic derecho en la carpeta Vista, agregar, elemento nuevo como se muestra en la siguiente ilustración.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 35. Agregar nuevo elemento a la carpeta Vista.

Seguido de eso se deberá seleccionar el tipo de archivo que se va a crear en la carpeta lo cual es un archivo del tipo Página (WPF), es un archivo de extensión XAML que se encuentra detallado en la sección del Marco Teórico, los dos archivos Login y ChatPage son del mismo tipo.

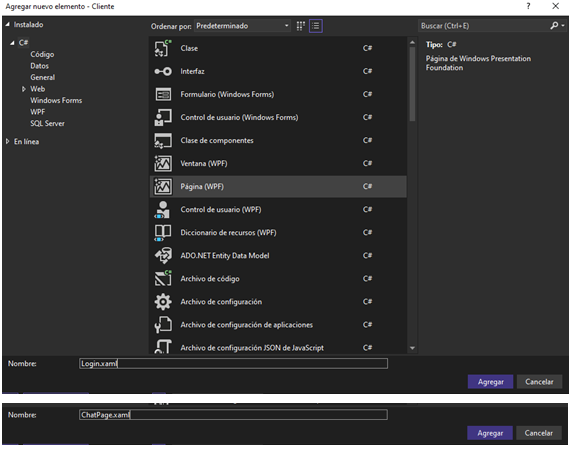


Figura 36. Selección del tipo de archivo y asignación de nombre a los archivos.

#### **CONTROLADOR**

Para agregar archivos a la carpeta Controlador se deberá hacer clic derecho sobre la carpeta Controlador, se deberá seleccionar agregar y elemento nuevo como se muestra en la siguiente ilustración.

Interfaz de usuario gráfica, Aplicación, Teams

Descripción generada automáticamente

Figura 37. Agregar nuevo elemento a la carpeta Controlador.

Una vez seleccionado nuevo elemento se deberá elegir el tipo de archivo y asignar un nombre a los archivos que se van a crear, en este caso son dos archivos los cuales son Chat y Client.

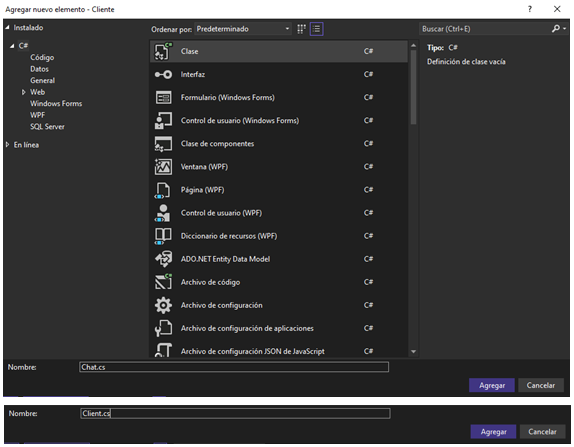


Figura 38. Selección del tipo de archivo y asignación de nombre.

#### **UTILS**

Para agregar un archivo a la carpeta Utils se deberá hacer clic derecho sobre la carpeta Utils, se deberá seleccionar agregar y elemento nuevo como se muestra en la siguiente ilustración.

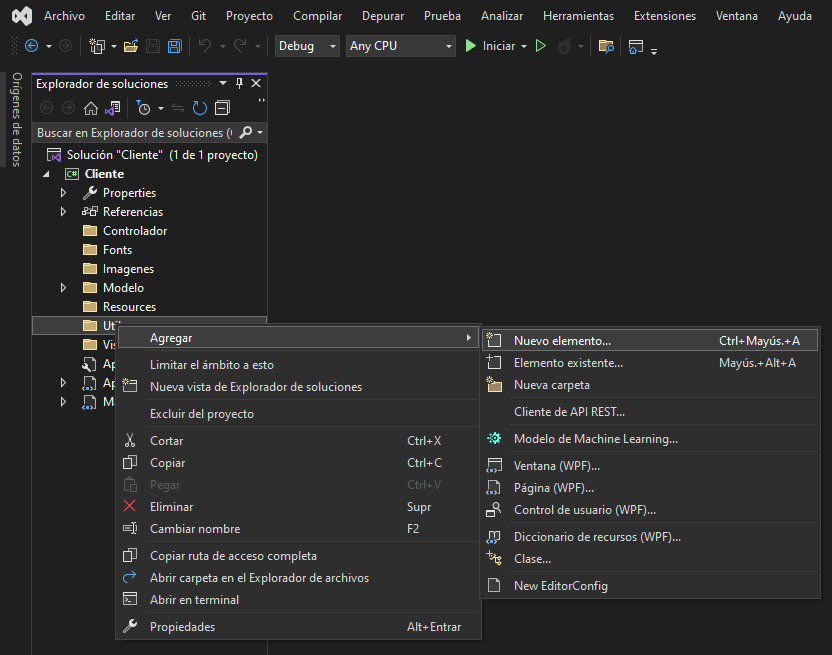


Figura 39. Agregar nuevo elemento a la carpeta Utils.

Seguido de eso se deberá elegir el tipo de archivo que se va a crear y asignarlo un nombre.

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 40. Selección del tipo de archivo y asignación de nombre.

#### **COMPONENTS**

Para agregar un archivo a la carpeta Components se deberá hacer clic derecho sobre la carpeta Components, se deberá seleccionar agregar y elemento nuevo como se muestra en la siguiente ilustración

Captura de pantalla de computadora

Descripción generada automáticamente

Figura 41. Agregar elementos nuevos a la carpeta Components.

Los archivos que se crearán en la carpeta Components son: el input para ingresar los mensajes y el botón, los archivos son del tipo Página (WPF).

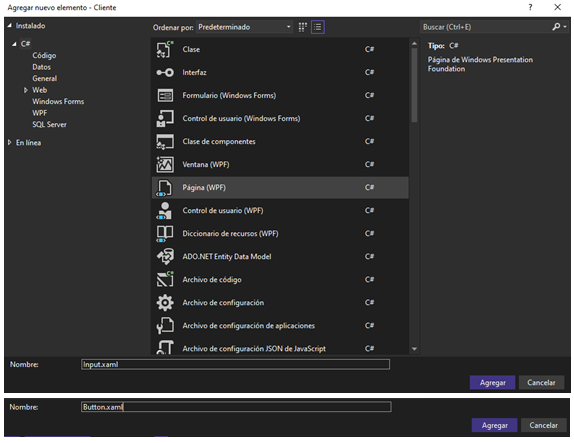


Figura 42. Selección del tipo de archivo y asignación de nombre.

#### **FONTS**

Para agregar las fuentes que se va a utilizar en el proyecto Cliente se requiere seguir los siguientes pasos.

1. Descargar las fuentes de letras para ellos se deberá ingresar al siguiente enlace.
   1. Fuente: Montserrat <https://fonts.google.com/specimen/Montserrat>

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 43. Fuente Montserrat

* 1. Fuente: Montserrat Alternates <https://fonts.google.com/specimen/Montserrat+Alternates>

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 44. Fuente Montserrat Alternates

Una vez ingresados a los siguientes enlaces se procede hacer clic en Download family y se descargará una carpeta .rar como se muestra en la siguiente ilustración.



Figura 45. Archivos descargados.

1. Extraer los archivos descargados.

Para ellos se deberá dirigir a la ruta donde se encuentran los archivos descargados, seguido de eso se deberá hacer clic derecho sobre cada una de las carpetas y Seleccionar en Extraer en…



Figura 46. Extraer archivos contenidos en las carpetas descargadas.

1. Seleccionar el tipo de fuente de cada una de las carpetas que se va a utilizar.

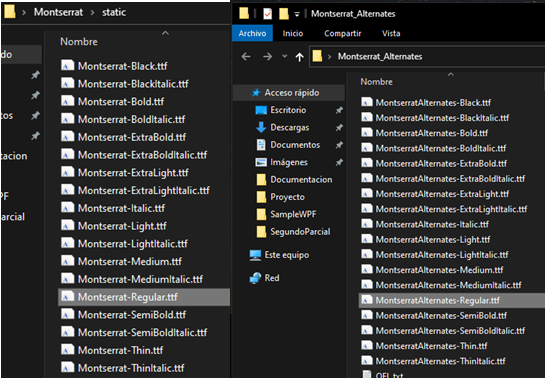


Figura 47. Selección del tipo de fuente de cada carpeta.

1. Copiar los archivos seleccionados a la carpeta Fonts del proyecto Cliente.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 48. Copiar los tipos de fuente en la carpeta Fonts.

1. Una vez copiados los archivos en la carpeta Fonts se procede a agregarlas al proyecto desde Visual Studio 2022, para ellos se deberá hacer clic derecho sobre la carpeta Fonts, agregar y elemento existente como se muestra en la siguiente ilustración.

Captura de pantalla de un celular

Descripción generada automáticamente

Figura 49. Agregar elemento existente a la carpeta Fonts.

Se abre una ventana emergente donde se mostrarán los archivos que deseamos agregar a la carpeta en este caso son las dos fuentes de letras que se va a utilizar en le proyecto como se muestra en la siguiente ilustración.

Captura de pantalla de un celular

Descripción generada automáticamente

Figura 50. Selección de archivos que se agregaran a la carpeta Fonts.

Finalmente se obtiene la siguiente estructura en la carpeta Fonts.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 51. Estructura de la carpeta Fonts.

#### **RESOURCES**

Agregamos las imágenes que se van a utilizar en el proyecto en la carpeta Resources.

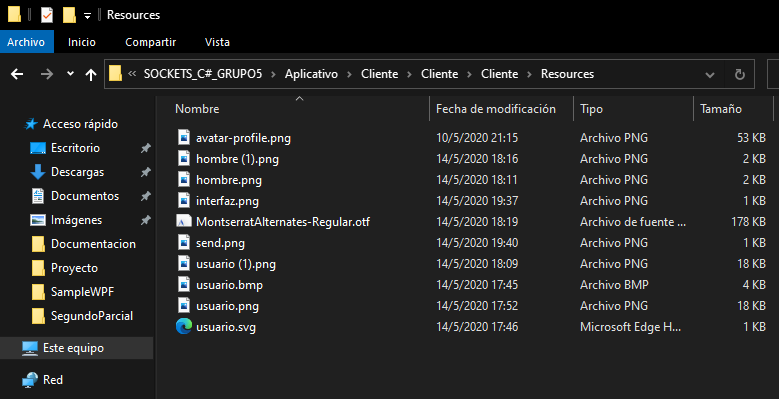


Figura 52. Agregar imágenes en la carpeta Resources.

Seguido de eso agregamos las imágenes al proyecto en Visual Studio 2022 para ello se hace clic derecho sobre la carpeta, agregar, elemento existente.

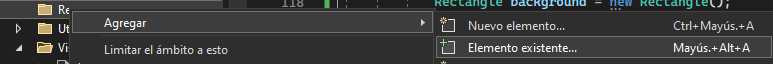


Figura 53. Agregamos elementos existentes.

Finalmente se realiza la creación del archivo de Recursos al proyecto que nos sirva para declarar los distintos tipos de archivos que se van a utilizar, para ello se hace clic derecho sobre el proyecto, agregar, elemento nuevo.

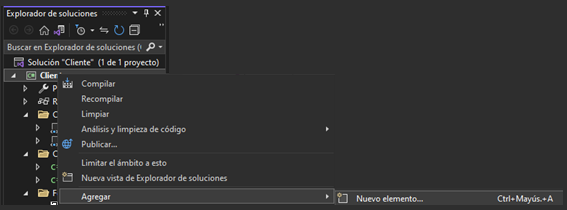


Figura 54. Agregar archivo de recursos.

Se deberá selecciona el tipo de archivo que se va a crear en este caso es de recurso.

Captura de pantalla de un celular

Descripción generada automáticamente

Figura 55. Selección del tipo de archivo y asignación de nombre.

Una vez creado el archivo de recurso se deberá agregar los archivos que se van a utilizar en el proyecto, las cuales serán las imágenes y las fuentes de letra, como se muestra en la siguiente ilustración.

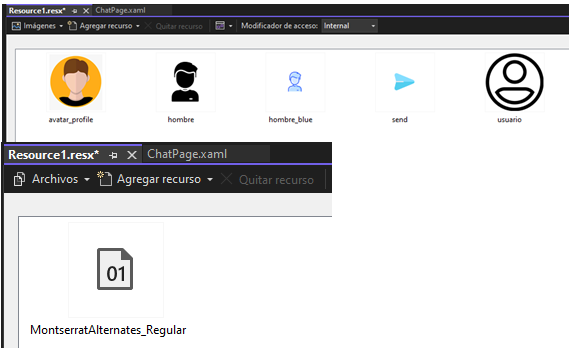


Figura 56. Asignación de recursos al archivo.

### **CODIFICACIÓN DEL PROYECTO CLIENTE**

#### **MODELO – USER**

Tabla 13. Codificación del archivo User.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Chat

{

[Serializable]

public class User

{

public string id;

public string nick;

public User(string id, string nick)

{

this.id = id;

this.nick = nick;

}

}

}

#### **MODELO – MESSAGE**

Tabla 14. Codificación del archivo Message.cs

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Chat

{

[Serializable]

public class Message

{

public User from;

public User to;

public string msg;

public Message(User from, User to, string msg)

{

this.from = from;

this.to = to;

this.msg = msg;

}

}

}

#### **CONTROLADOR – CHAT**

Tabla 15. Codificación del archivo Chat.cs

using System.Collections.Generic;

using Chat;

namespace Cliente.Chat

{

class ChatManager

{

LinkedList<User> users;

LinkedList<Message> messages;

User current;

public ChatManager(User current)

{

this.users = new LinkedList<User>();

this.messages = new LinkedList<Message>();

this.current = current;

}

public bool AddUser(User u)

{

LinkedListNode<User> currentUser = users.Find(u);

if (currentUser == null && this.current.id != u.id)

{

this.users.AddLast(u);

return true;

}

else

{

return false;

}

}

public void AddMessage(Message m)

{

this.messages.AddLast(m);

}

public LinkedList<Message> GetMessages(User from)

{

LinkedList<Message> tmpMSG = new LinkedList<Message>();

LinkedListNode<Message> currentMSG = this.messages.First;

while (currentMSG != null)

{

if (currentMSG.Value.from.id == from.id || currentMSG.Value.to.id == from.id)

{

tmpMSG.AddLast(currentMSG.Value);

}

currentMSG = currentMSG.Next;

}

return tmpMSG;

}

public LinkedList<User> GetUsers()

{

return this.users;

}

}

}

#### **CONTROLADOR – CLIENT**

Tabla 16. Codificación del archivo Client.cs

using System;

using System.Collections.Generic;

using System.Net.Sockets;

using System.Net;

using System.Threading;

using Serialization;

using Chat;

namespace Cliente.Chat

{

public class Client

{

public delegate void UpdateDelegate(object o);

public UpdateDelegate onUpdate;

User u;

ChatManager chat;

readonly IPHostEntry host;

readonly IPAddress addr;

readonly IPEndPoint endPoint;

readonly Socket socket;

Thread listenThread;

public Client(string username, UpdateDelegate onUpdate)

{

this.onUpdate = onUpdate;

this.u = new User(Guid.NewGuid().ToString("N"), username);

this.chat = new ChatManager(u);

try

{

host = Dns.GetHostEntry("localhost");

addr = host.AddressList[0];

endPoint = new IPEndPoint(addr, 4404);

socket = new Socket(endPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

public void Start()

{

this.socket.Connect(endPoint);

this.Send(this.socket, this.u);

listenThread = new Thread(this.Listen);

listenThread.SetApartmentState(ApartmentState.STA);

listenThread.Start();

}

private void Listen()

{

object received;

while (true)

{

received = this.Receive(this.socket);

if (received is Message)

{

this.AddMessage((Message)received);

}

else if (received is User)

{

this.AddUser((User)received);

}

}

}

private void AddUser(User user)

{

if (user.id != this.u.id && this.chat.AddUser(user))

onUpdate(user);

}

private void AddMessage(Message m)

{

this.chat.AddMessage(m);

onUpdate(m);

}

public LinkedList<Message> GetMessages(User u)

{

return chat.GetMessages(u);

}

public LinkedList<User> GetUsers()

{

return chat.GetUsers();

}

public Message SendMessage(string msg, User to)

{

Message m = new Message(this.u, to, msg);

this.Send(this.socket, m);

this.AddMessage(m);

return m;

}

public Message SendFile(string file, User to)

{

Message fl = new Message(this.u, to, file);

this.Send(this.socket, fl);

return fl;

}

private void Send(Socket s, object o)

{

byte[] buffer = new byte[1024];

byte[] obj = BinarySerialization.Serializate(o);

Array.Copy(obj, buffer, obj.Length);

s.Send(buffer);

}

private object Receive(Socket s)

{

byte[] buffer = new byte[1024];

s.Receive(buffer);

return BinarySerialization.Deserializate(buffer);

}

}

}

#### **UTILS – SERIALIZATION**

Tabla 17. Codificación del archivo Serialization.cs

using System;

using System.IO;

using System.Reflection;

using System.Runtime.Serialization;

using System.Runtime.Serialization.Formatters.Binary;

namespace Serialization

{

class BinarySerialization

{

public static byte[] Serializate(object toSerializate){

MemoryStream memory = new MemoryStream();

BinaryFormatter formatter = new BinaryFormatter();

formatter.Serialize(memory, toSerializate);

return memory.ToArray();

}

public static object Deserializate(byte[] data)

{

MemoryStream memory = new MemoryStream(data);

BinaryFormatter formatter = new BinaryFormatter();

formatter.Binder = new CurrentAssemblyDeserializationBinder();

return formatter.Deserialize(memory);}}

public class CurrentAssemblyDeserializationBinder : SerializationBinder

{

public override Type BindToType(string assemblyName, string typeName)

{

return Type.GetType(String.Format("{0}, {1}", typeName, Assembly.GetExecutingAssembly().FullName));

}

}

#### **COMPONENTS – INPUT (Interfaz gráfica)**

Tabla 18. Codificación del archivo Input.xaml

<UserControl x:Class="Cliente.Components.Input"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:Cliente.Components"

mc:Ignorable="d"

d:DesignHeight="40" d:DesignWidth="800">

<UserControl.Resources>

<Style TargetType="TextBox" x:Key="Input">

<Setter Property="BorderThickness" Value="0"></Setter>

</Style>

</UserControl.Resources>

<Border Grid.Column="0" Grid.Row="1" CornerRadius="15" BorderThickness="1" BorderBrush="#aeaeae" Padding="5" Margin="5" Background="#ffffff">

<TextBox DockPanel.Dock="Left" Style="{StaticResource Input}" Name="textInput" KeyUp="TextInput\_KeyUp"/>

</Border>

</UserControl>

#### **COMPONENTS – INPUT (Lógica de los elementos)**

Tabla 19. Codificación del archivo Input.xaml.cs

using System;

using System.Windows.Controls;

using System.Windows.Input;

namespace Cliente.Components

{

public partial class Input : UserControl

{

public EventHandler onEnter;

public Input()

{

InitializeComponent();

}

public void TextInput\_KeyUp(object sender, KeyEventArgs e)

{

if (e.Key == Key.Enter && onEnter != null)

{

onEnter(sender, e);

}

}

}

}

#### **COMPONENTS – BUTTON (Interfaz gráfica)**

Tabla 20. Codificación del archivo Button.xaml

<UserControl x:Class="Cliente.Components.Button"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:Cliente.Components"

mc:Ignorable="d"

d:DesignHeight="25" d:DesignWidth="1200">

<UserControl.Resources>

<Style TargetType="{x:Type Button}" x:Key="button">

<Setter Property="Background" Value="#42a5f5"></Setter>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="{x:Type Button}">

<Border Background="{TemplateBinding Background}" CornerRadius="15">

<ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center" Margin="5"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Background" Value="#1e88e5"/>

<Setter Property="Foreground" Value="yellow"/>

<Setter Property="Cursor" Value="Hand"/>

</Trigger>

</Style.Triggers>

</Style>

</UserControl.Resources>

<Button Content="{Binding Text}" Foreground="white" BorderThickness="0" FontSize="16" Style="{StaticResource button}" Click="Click"></Button>

</UserControl>

#### **COMPONENTS – BUTTON (Lógica de los elementos)**

Tabla 21. Codificación del archivo Button.xaml.cs

using System;

using System.Windows;

using System.Windows.Controls;

namespace Cliente.Components

{

public partial class Button : UserControl

{

public EventHandler clickEvent { set; get; }

public string Text { get; set; }

public Button()

{

InitializeComponent();

this.DataContext = this;

}

private void Click(object sender, RoutedEventArgs e)

{

if (this.clickEvent != null) clickEvent(sender, e);

}

}

}

#### **COMPONENTS – INPUTFILE**

Tabla 22. Codificación del archivo Inputfile.xaml

<UserControl x:Class="Cliente.Components.InputFile"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:Cliente.Components"

mc:Ignorable="d"

d:DesignHeight="25" d:DesignWidth="1200">

<UserControl.Resources>

<Style TargetType="{x:Type Button}" x:Key="button">

<Setter Property="Background" Value="#42a5f5"></Setter>

<Setter Property="Template">

<Setter.Value>

<ControlTemplate TargetType="{x:Type Button}">

<Border Background="{TemplateBinding Background}" CornerRadius="15">

<ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center" Margin="5"/>

</Border>

</ControlTemplate>

</Setter.Value>

</Setter>

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Background" Value="#1e88e5"/>

<Setter Property="Foreground" Value="yellow"/>

<Setter Property="Cursor" Value="Hand"/>

</Trigger>

</Style.Triggers>

</Style>

</UserControl.Resources>

<Button Content="Archivo" Foreground="white" BorderThickness="0" FontSize="16" Style="{StaticResource button}" Click="OpenFileButton\_Click"></Button>

</UserControl>

#### **COMPONENTS – INPUTFILE (Lógica de los elementos)**

Tabla 23. Codificación del archivo InputFile.xaml.cs

using Microsoft.Win32;

using System;

using System.Text;

using System.Windows;

using System.Windows.Controls;

namespace Cliente.Components

{

public partial class InputFile : UserControl

{

public EventHandler onSelected;

public InputFile()

{

InitializeComponent();

}

private void OpenFileButton\_Click(object sender, RoutedEventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog();

if (openFileDialog.ShowDialog() == true){

string filePath = openFileDialog.FileName;

byte[] filePathByte = Encoding.UTF8.GetBytes(filePath);

onSelected(filePathByte, e);}

}

}

}

#### **VISTA – LOGIN (Interfaz gráfica)**

Tabla 24. Codificación del archivo InputFile.xaml

<Page x:Class="Cliente.Login"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:Cliente.Vista"

xmlns:uc="clr-namespace:Cliente.Components"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800" Title="Login"

FontFamily="Fonts/MontserratAlternates-Regular.otf #Montserrat Alternates">

<Grid >

<Grid.ColumnDefinitions>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label Grid.Column="0" Grid.Row="0" Margin="5" FontSize="16">Usuario</Label>

<uc:Input Grid.Row="1" FontSize="16" x:Name="username"/>

<uc:Button Grid.Column="0" Grid.Row="2" Margin="5" Text="Ingresar" x:Name="btn">

</uc:Button>

</Grid>

</Page>

#### **VISTA – LOGIN (Lógica de los elementos)**

Tabla 25. Codificación del archivo Login.xaml.cs

using Cliente.Chat;

using System;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Navigation;

namespace Cliente

{

public partial class Login : Page

{

public Login()

{

InitializeComponent();

btn.clickEvent = StartChat;

}

private void StartChat(object sender, EventArgs e)

{

string user = username.textInput.Text;

Vista.ChatPage chatPage;

Client c;

if (user.Trim() != "")

{

chatPage = new Vista.ChatPage();

c = new Client(user, chatPage.UpdateUI);

c.Start();

chatPage.client = c;

NavigationService.Navigate(chatPage);

}

else

MessageBox.Show("Ingrese un usuario");

}

}

}

#### **VISTA – CHATPAGE (Interfaz gráfica)**

Tabla 26. Codificación del archivo ChatPage.xaml

<Page x:Class="Cliente.Vista.ChatPage"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:Cliente.Vista"

xmlns:uc="clr-namespace:Cliente.Components"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800"

Title="ChatPage">

<Page.Resources>

<Style TargetType="ButtonBase" x:Key="btn">

<Style.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Background" Value="Transparent"></Setter>

</Trigger>

</Style.Triggers>

</Style>

<Style TargetType="Border" x:Key="message-primary">

<Setter Property="Background" Value="#2196f3"></Setter>

<Setter Property="CornerRadius" Value="20,20,0,20"></Setter>

<Setter Property="Margin" Value="50,20,20,20"></Setter>

<Setter Property="MaxWidth" Value="{Binding ActualWidth, ElementName=Chat\_Content}"></Setter>

</Style>

<Style TargetType="TextBlock" x:Key="message-primary-content">

<Setter Property="Foreground" Value="#ffffff"></Setter>

<Setter Property="TextWrapping" Value="Wrap"></Setter>

<Setter Property="Margin" Value="20"></Setter>

<Setter Property="TextAlignment" Value="Right"></Setter>

</Style>

<Style TargetType="Border" x:Key="message-secondary">

<Setter Property="Background" Value="#0D000000"></Setter>

<Setter Property="CornerRadius" Value="20,20,20,0"></Setter>

<Setter Property="Margin" Value="20,20,50,20"></Setter>

<Setter Property="MaxWidth" Value="{Binding ActualWidth, ElementName=Chat\_Content}"></Setter>

</Style>

<Style TargetType="TextBlock" x:Key="message-secondary-content">

<Setter Property="Foreground" Value="#000000"></Setter>

<Setter Property="TextWrapping" Value="Wrap"></Setter>

<Setter Property="Margin" Value="20"></Setter>

<Setter Property="TextAlignment" Value="Left"></Setter>

</Style>

</Page.Resources>

<DockPanel>

<Grid DockPanel.Dock="Left" Width="200" Name="UsersContainer" Background="#eceff1" >

<Grid.ColumnDefinitions>

<ColumnDefinition Width="60px"/>

<ColumnDefinition Width="140px"/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label Grid.Column="0" Grid.ColumnSpan="2" Grid.Row="0" HorizontalAlignment="Center" FontSize="14">Usuarios</Label>

</Grid>

<Label DockPanel.Dock="Top" FontSize="16" Name="Username"></Label>

<DockPanel DockPanel.Dock="Bottom" Height="49">

<Button DockPanel.Dock="Right" Background="Transparent" BorderThickness="0" Style="{StaticResource btn}">

<Image Width="24" Source="../images/send.png"></Image>

</Button>

<uc:Input x:Name="msgInput" Width="424" Height="40"/>

<uc:InputFile Width="122" Height="34" x:Name="fileInput" />

</DockPanel>

<ScrollViewer CanContentScroll="False" Width="Auto" VerticalScrollBarVisibility="Auto" Name="Chat\_Content" Height="381">

<StackPanel Name="MessagesPanel">

</StackPanel>

</ScrollViewer>

</DockPanel>

</Page>

#### **VISTA – CHATPAGE (Lógica de los elementos)**

Tabla 27. Codificación del archivo ChatPage.xaml.cs

using Cliente.Chat;

using Chat;

using System;

using System.Collections.Generic;

using System.Text;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Interop;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Net;

using Microsoft.Win32;

using System.Windows.Shapes;

namespace Cliente.Vista

{

public partial class ChatPage : Page

{

public Client client;

User \_context;

User context

{

set

{

if (value != null)

{

\_context = value;

this.ShowMessages();

this.Username.Content = value.nick;

}

}

get { return \_context; }

}

public ChatPage()

{

InitializeComponent();

context = null;

msgInput.onEnter = SendMessage;

fileInput.onSelected = SendFile;

}

private void SendMessage(object sender, EventArgs e)

{

Message msg;

string message = ((TextBox)sender).Text;

if (this.context != null && message.Trim() != "")

{

msg = this.client.SendMessage(message, this.context);

this.addMessage(msg);

}

((TextBox)sender).Text = "";

}

private void SendFile(object sender, EventArgs e)

{

Message msg;

byte[] fileArray = (byte[])sender;

string file = Encoding.UTF8.GetString(fileArray);

if (this.context != null && file.Trim() != "")

{

msg = this.client.SendFile(file, this.context);

this.downloadFile(msg);

}

}

private async void downloadFile(Message msg)

{

using (WebClient webClient = new WebClient())

{

try

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

int lastSlashIndex = msg.msg.LastIndexOf('\\');

string fileName = msg.msg.Substring(lastSlashIndex + 1);

saveFileDialog.FileName = fileName;

if (MessageBox.Show("Descargar ..." + fileName,

"¿ " + msg.to.nick + " deseas recibir archivo de " + msg.from.nick + " ?",

MessageBoxButton.YesNo) == MessageBoxResult.Yes)

{

saveFileDialog.ShowDialog();

string pathLocal = saveFileDialog.FileName;

await webClient.DownloadFileTaskAsync(new Uri(msg.msg), pathLocal);

saveFileDialog.Title = "Seleccione la ruta de descarga";

MessageBox.Show("Archivo descargado");

}

else

{

msg.msg = "Archivo rechazado";

this.addMessage(msg);

MessageBox.Show("Archivo rechazado");

}

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

}

}

private void ShowMessages()

{

User u = this.context;

LinkedList<Message> current = this.client.GetMessages(context);

LinkedListNode<Message> msg = current.First;

MessagesPanel.Children.Clear();

for (int i = 0; i < current.Count; i++)

{

this.addMessage(msg.Value);

msg = msg.Next;

}

}public void UpdateUI(object o)

{

Dispatcher.Invoke(new Action(() =>

{

if (o is User)

{

this.addUser((User)o);

}

else if (o is Message)

{

if (this.context != null && ((Message)o).from.id == this.context.id)

this.addMessage((Message)o);

}

}));

}

private void addMessage(Message msg)

{

Style containerStyle;

Style contentStyle;

if (msg.from.id == this.context.id)

{

containerStyle = this.FindResource("message-secondary") as Style;

contentStyle = this.FindResource("message-secondary-content") as Style;

}

else

{

containerStyle = this.FindResource("message-primary") as Style;

contentStyle = this.FindResource("message-primary-content") as Style;

}

Border msgContainer = new Border();

msgContainer.Style = containerStyle;

TextBlock msgContent = new TextBlock();

msgContent.Text = msg.msg;

msgContent.Style = contentStyle;

msgContainer.Child = msgContent;

MessagesPanel.Children.Add(msgContainer);

}

private void addUser(User u)

{

BitmapSource source = Imaging.CreateBitmapSourceFromHBitmap(Resource1.hombre\_blue.GetHbitmap(), IntPtr.Zero, Int32Rect.Empty, BitmapSizeOptions.FromEmptyOptions());

Rectangle background = new Rectangle();

background.Fill = new SolidColorBrush((Color)ColorConverter.ConvertFromString("#eceff1"));

Border border = new Border();

border.CornerRadius = new CornerRadius(30.0);

border.Height = 32;

border.Width = 32;

border.Background = new ImageBrush(source);

Label name = new Label();

if (u.nick.Length > 10)

name.Content = u.nick.Substring(0, 10) + "...";

else

name.Content = u.nick;

name.VerticalAlignment = VerticalAlignment.Center; MouseEventHandler mouseEnter = new MouseEventHandler((sender, e) => { this.ChangeBackground(background, "#0D000000"); Mouse.OverrideCursor = Cursors.Hand; });

MouseEventHandler mouseLeave = new MouseEventHandler((sender, e) => { this.ChangeBackground(background, "#eceff1"); Mouse.OverrideCursor = Cursors.Arrow; });

MouseButtonEventHandler mouseLeftClick = new MouseButtonEventHandler((sender, o) => { this.context = u; });

background.MouseEnter += mouseEnter;

background.MouseLeave += mouseLeave;

background.MouseLeftButtonUp += mouseLeftClick;

border.MouseEnter += mouseEnter;

border.MouseLeave += mouseLeave;

border.MouseLeftButtonUp += mouseLeftClick;

name.MouseEnter += mouseEnter;

name.MouseLeave += mouseLeave;

name.MouseLeftButtonUp += mouseLeftClick;

RowDefinition row = new RowDefinition();

row.Height = new GridLength(40);

UsersContainer.RowDefinitions.Add(row);

MouseEventHandler mouseEnter = new MouseEventHandler((sender, e) => { this.ChangeBackground(background, "#0D000000"); Mouse.OverrideCursor = Cursors.Hand; });

MouseEventHandler mouseLeave = new MouseEventHandler((sender, e) => { this.ChangeBackground(background, "#eceff1"); Mouse.OverrideCursor = Cursors.Arrow; });

MouseButtonEventHandler mouseLeftClick = new MouseButtonEventHandler((sender, o) => { this.context = u; });

background.MouseEnter += mouseEnter;

background.MouseLeave += mouseLeave;

background.MouseLeftButtonUp += mouseLeftClick;

border.MouseEnter += mouseEnter;

border.MouseLeave += mouseLeave;

border.MouseLeftButtonUp += mouseLeftClick;

name.MouseEnter += mouseEnter;

name.MouseLeave += mouseLeave;

name.MouseLeftButtonUp += mouseLeftClick;

RowDefinition row = new RowDefinition();

row.Height = new GridLength(40);

UsersContainer.RowDefinitions.Add(row);

UsersContainer.Children.Add(background);

UsersContainer.Children.Add(border);

UsersContainer.Children.Add(name);

Grid.SetRow(background, UsersContainer.RowDefinitions.Count - 1);

Grid.SetColumn(background, 0);

Grid.SetColumnSpan(background, 2);

Grid.SetColumn(border, 0);

Grid.SetRow(border, UsersContainer.RowDefinitions.Count - 1);

Grid.SetColumn(name, 1);

Grid.SetRow(name, UsersContainer.RowDefinitions.Count - 1);

}

private void ChangeBackground(object sender, string color)

{

Rectangle r = (Rectangle)sender;

r.Fill = new SolidColorBrush((Color)ColorConverter.ConvertFromString(color));

}

private void msgInput\_Loaded(object sender, RoutedEventArgs e)

{

}

}

#### **ARCHIVO – MainWindows (Interfaz gráfica)**

Tabla 28. Codificación del archivo MainWindows.xaml

<Window x:Class="Cliente.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:Cliente"

mc:Ignorable="d"

Title="Grupo 5 Sockets" Height="450" Width="800">

<Frame Source="Vista/Login.xaml" x:Name="NavigationFrame" NavigationUIVisibility="Hidden"/>

</Window>

#### **ARCHIVO – MainWindows (Lógica de los elementos)**

Tabla 29. Codificación del archivo MainWindows.xaml.cs

using System.Windows;

namespace Cliente

{

public partial class MainWindow : Window

{

public MainWindow()

{

InitializeComponent();

}

}

}

## EJECUCIÓN DEL PROYECTO

Considerando que el proyecto realizado contiene dos proyectos los cuales son el Cliente y el Servidor se procederá primero a la ejecución del proyecto Servidor, puesto que se requiere de este servicio para poder registrarnos como usuarios en el Cliente.

Para ejecutar el proyecto Servidor se tiene que hacer clic sobre iniciar como se muestra en la siguiente ilustración.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 57. Ejecución del proyecto Servidor

Una vez que se levanta el proyecto servidor se mostrará una ventana de consola donde se podrá ver la actividad que se encuentra ejecutándose en el cliente.

Texto

Descripción generada automáticamente

Figura 58. Servidor esperando la conexión de los usuarios.

Una vez que tenemos el servidor en línea esperando la conexión de los usuarios, procedemos a ejecutar el proyecto cliente, para ello nos dirigimos al proyecto y le damos clic en la función de iniciar como se muestra en la siguiente ilustración.

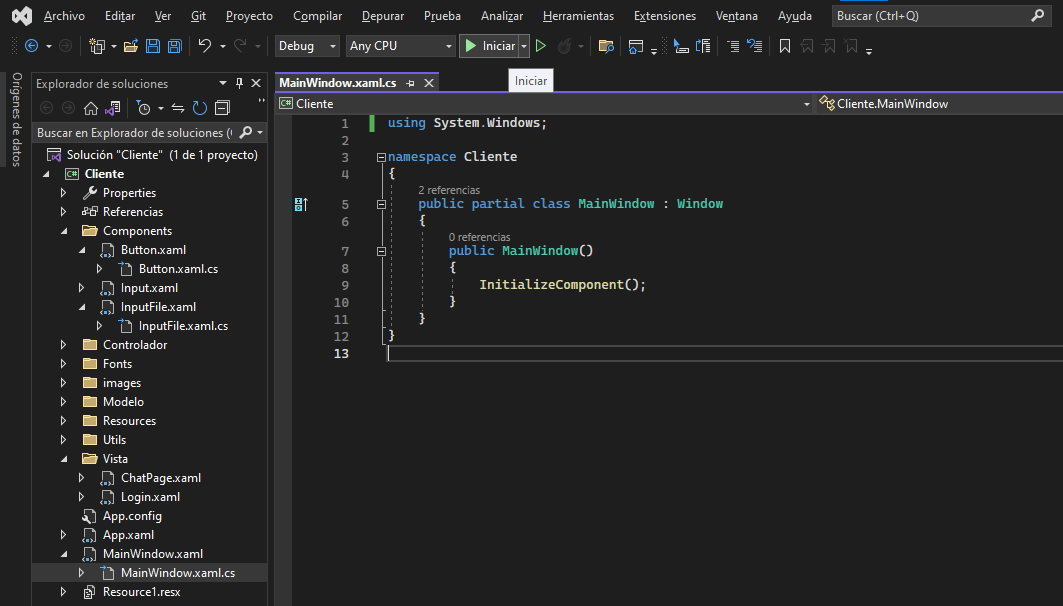


Figura 59. Ejecución del proyecto Cliente

Se muestra la siguiente ilustración donde se pedirá el nombre del usuario que desea conectarse a la aplicación de mensajería.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 60. Vista principal de la aplicación de Chat.

Se ingresa el dato solicitado el cual el usuario tomará el nombre de Kevin y hacemos clic en siguiente.

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Figura 61. Ingreso del nombre del primer usuario.

Una vez ingresado el nombre del usuario se accede a la ventana de mensajería, pero puesto que la aplicación es un chat se requiere de al menos dos usuarios.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Figura 62. Ventana principal del Chat del usuario Kevin.

Se procede a realizar el registro de un segundo usuario para poder mantener una conversación entre los dos usuarios registrados, para ello hay que acceder a una ruta donde se almacena el .exe del proyecto, el segundo usuario lo vamos a registrar con el nombre de Steven.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Figura 63. Ingreso del nombre del segundo usuario.

Una vez registrado el segundo usuario se puede ver en la pantalla de los usuarios que les aparece en línea los usuarios registrados, y con los que pueden mantener una comunicación como se muestra en la siguiente ilustración.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 64. Lista de usuarios con los cuales se puede establecer una comunicación.

Se procede a realizar una comunicación por medio de mensajes entre el usuario Kevin y Steven.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 65. Mensaje del usuario Steven a Kevin.

En la siguiente ilustración se puede visualizar la comunicación entre los dos usuarios.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Figura 66. Comunicación entre los usuarios Steven y Kevin.

Además, se puede visualizar la actividad en el servidor.

Texto

Descripción generada automáticamente

Figura 67. Actividad del cliente.

Se procede a realizar el envío de archivos por medio del Chat entre el usuario Steven y Kevin.

Para ello se deberá seleccionar la opción de Archivo como se muestra en la siguiente ilustración.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Figura 68. Elección de la función archivo.

Se deberá seleccionar el archivo que se requiere enviar al otro usuario, puede ser un archivo de

Audio, video, imágenes, documentos o cualquier tipo de archivo que el usuario desee compartir, en este caso se seleccionará un archivo .sql

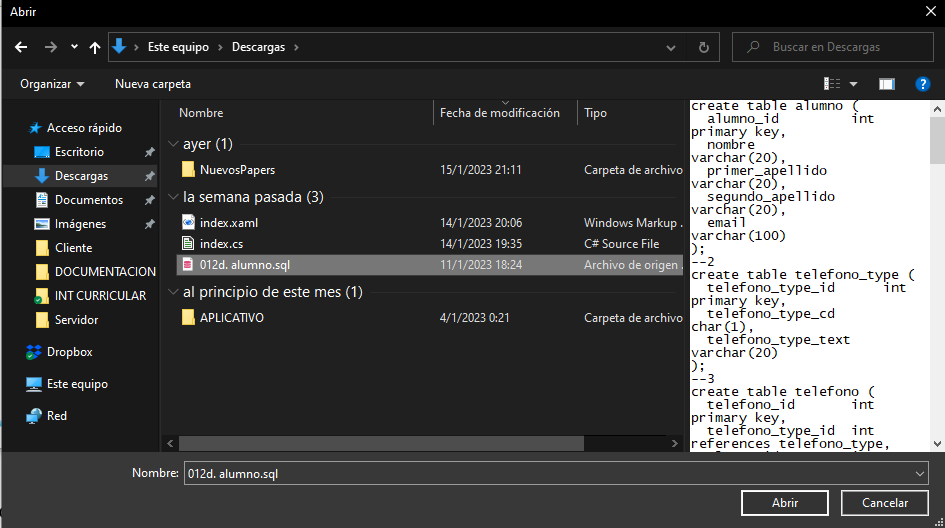


Figura 69. Selección del archivo para enviar.

Una vez seleccionado el archivo se procede a elegir la opción de archivo, y de esta forma le llegará una notificación al usuario que se le quiere enviar el archivo si desea descargar el archivo que le acaba de llegar.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 70. Notificación de descarga del archivo que Steven envía a Kevin.

En este caso Kevin aceptará el archivo que le llega por parte de Kevin y se abrirá la siguiente ventana donde se deberá seleccionar la ruta de descarga del archivo.

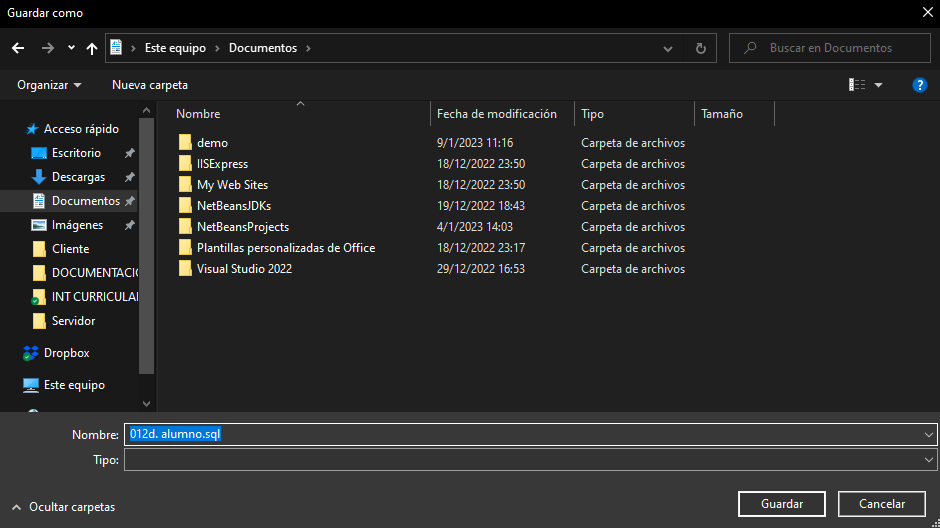


Figura 71. Elección de ruta de descarga del archivo recibido

Se hace clic en la opción guardar, se mostrará una notificación de Archivo descargado y se podrá visualizar la descarga en la ruta seleccionada anteriormente.

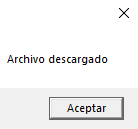


Figura 72. Notificación de archivo descargado.

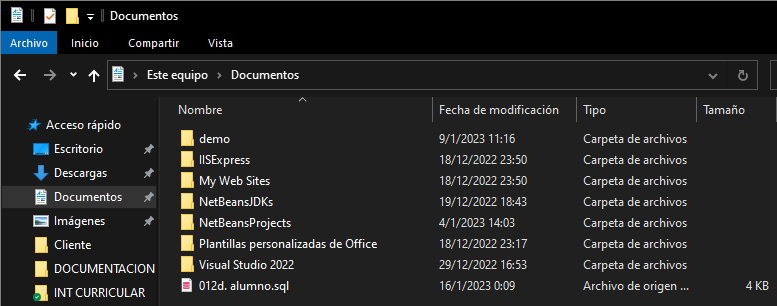


Figura 73. Visualización del archivo descargado.

También existe la posibilidad de que el usuario no acepte descargar el archivo que se lo esta enviando para ello esta en la capacidad de rechazar el archivo y de esta forma se le notificará al usuario que lo envío que su archivo fue rechazado.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Figura 74. Notificación y envió de mensaje al usuario de Archivo rechazado.

# CONCLUSIONES

* Los sockets nos permiten implementar una arquitectura cliente – servidor donde la comunicación es iniciada por uno de los procesos el cual es denominado “Cliente” y finalmente se espera al segundo proceso inicie la comunicación por tal razón es conocido como servidor.
* Un socket nos permite el intercambio de información entre procesos en la misma máquina o en una red de tal manera que se puede distribuir el trabajo y tener el acceso a datos centralizados.
* Un socket es un proceso o hilo que existe en la máquina del cliente y servidor, la cual sirve como instancia para que un programa servidor y el cliente tengan acceso a la información la misma que es transmitida por varias capas de red.

# RECOMENDACIONES

* Antes de comenzar con el desarrollo de proyectos haciendo uso de sockets es importante el conocer los hilos.
* Para el desarrollo de la práctica es importante el conocer que es una arquitectura cliente – servidor, además, del Modelo Vista Controlador.

# REFERENCIAS

* Rivera, J. (s.f.). Sockets en Java. Obtenido de <http://pedrobeltrancanessabiblioteca.weebly.com/uploads/1/2/4/0/12405072/mi_manual_sockets.pdf>
* ULG. (s.f.). Socket programming. Obtenido de Complement for the programming assignment: <http://www.montefiore.ulg.ac.be/~hiard/Java%20Sockets.pdf>
* <https://cruzado.info/tutojava/V_2.htm>
* <https://www.youtube.com/watch?v=L0Y6hawPB-E&t=4s>
* Vila, J. (s.f.). Sockets en Java Sockets en J. Obtenido de https://poliformat.upv.es/access/content/group/OCW\_6069\_2008/T2.- Comunicaci%C3%B3n%20I%3A%20del%20C\_S%20al%20modelo%20de%20objetos/T ecnolog%C3%ADa%20JAVA/Java-sockets.pdf
* "TCP/IP Sockets in C: Practical Guide for Programmers" de Michael J. Donahoo y Kenneth L. Calvert.
* "Unix Network Programming" de W. Richard Stevens.
* "Windows Sockets Network Programming" de Bob Quinn y Dave Shute.
* https://www.infor.uva.es/~fdiaz/sd/doc/java.net.pdf
* <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/>
* <https://docs.microsoft.com/en-us/xamarin/essentials/>