-ejecutar programa paso a paso y apuntar en un papel

git clone https://github.com/j5anchez/PSPremix

git clone https://github.com/j5anchez/PSP03a

git clone https://github.com/j5anchez/PSP03b

git clone https://github.com/j5anchez/PSP04

git clone https://github.com/j5anchez/PSP05

git init

git add .

git commit -m "initial commit"

git remote add origin https://github.com/j5anchez/PSPremix.git

git push -u origin master

git push origin master

git config --global user.email "josebsanchez@birt.eus"

git config --global user.name "j5anchez"

asyn…….2

Juegoloteria(servidor)….4

fotoClienteServidor(Cliente)…7

fotoClienteServidor(Servidor)…10

servidorFTP(cliente)….14

servidorFTP(servidor)….17

passEncriptadas…..21

---

async

```
Thread t = new Thread(Tarea1); //Hemos creado un nuevo hilo

private static void Tarea1()
        {

        }
```

```
         Thread thread = new Thread(ReadFromServer);
                 thread.Start();    private void ReadFromServer()
         {

         }
```

```
using System;
using System.Collections.Concurrent;
using System.Threading.Tasks;

namespace AlquilerBicicletas
{
    class Program
    {
        static void Main(string[] args)
        {
            BlockingCollection<int> stockBicis = new BlockingCollection<int>(100);
            BlockingCollection<int> stockBiciSecundario = new BlockingCollection<int>(100);

            /*Tiene que haber 3 task:
             * 1. Productor: Empresa de Bicicletas
             * 2. Consumidor y productor (alquila y devuelve bicicletas): Zona Gros
             * 3. Consumidor y productor (alquila y devuelve bicicletas): Zona Amara
            */

            //Productor, empresa que compra bicicletas.

            Task compraBici = Task.Run(() =>
            {
                int bici = 0;
                bool maxAlmacen = false;
```

```
    while (!maxAlmacen)
    {
        stockBicis.Add(bici);
        Console.WriteLine("La empresa de bicis ha comprado la bicicleta{0} y la tiene en el almacén principal.", bici);
        bici++;
        //Thread.Sleep(100);
        if (bici == 200)
        {
            maxAlmacen = true;
        }
    }
    stockBicis.CompleteAdding();
    Console.WriteLine("Cierre de almacén. Nadie podrá depositar bicicletas en dicho almacén", bici);
});

Task ZonaGros = Task.Run(() =>
{
    while (!stockBicis.IsCompleted)
    {
        int bici = -1;

        try
        {
            bici = stockBicis.Take();
        }
        catch (InvalidOperationException)
        {
            Console.WriteLine("Error en Gros: ha habido problemas para alquilar bici.");
        }

        if (bici != -1)
        {
            Console.WriteLine("Un usuario en la zona Gros ha alquilado la bicicleta{0}.", bici);
        }

        if (!stockBicis.IsAddingCompleted)
        {
            if (bici % 3 == 0)
            {
                stockBicis.Add(bici);
                Console.WriteLine("Un usuario en la zona Gros ha devuelto la bicicleta{0}.", bici);
            }
        }
        else
        {
            if (bici % 3 == 0)
            {
                Console.WriteLine("El almacén principal está completo, se despositarán las bicis en el secundario.");
                stockBiciSecundario.Add(bici);
                Console.WriteLine("Un usuario en la zona Gros ha devuelto la bicicleta{0} al segundo almacén.", bici);
            }
        }
    }
    Console.WriteLine("En zona Gros no hay más bicis en el almacén.");
});

Task ZonaAmara = Task.Run(() =>
{
    int bici = -1;
```

```
            while (!stockBicis.IsCompleted)
            {
                try
                {
                    bici = stockBicis.Take();
                }
                catch (InvalidOperationException)
                {
                    Console.WriteLine("Error en Amara: ha habido problemas para alquilar bici.");
                }

                if (bici != -1)
                {
                    Console.WriteLine("Un usuario en la zona Amara ha alquilado la bicicleta{0}.", bici);
                }

                if (!stockBicis.IsAddingCompleted)
                {
                    if (bici % 5 == 0)
                    {
                        stockBicis.Add(bici);
                        Console.WriteLine("Un usuario en la zona Amara ha devuelto la bicicleta{0}.", bici);
                    }
                }
               else
                {
                    if (bici % 5 == 0)
                    {
                        Console.WriteLine("El almacén principal está completo, se despositarán las bicis en el secundario.");
                        stockBiciSecundario.Add(bici);
                        Console.WriteLine("Un usuario en la zona Amara ha devuelto la bicicleta{0} al segundo almacén.", bici);
                    }
                }

            }
            Console.WriteLine("En zona Amara no hay más bicis en el almacén.");
        });

        compraBici.Wait();
        ZonaGros.Wait();
        ZonaAmara.Wait();

        Console.WriteLine("El stock sobrante es {0} bicis.", stockBiciSecundario.Count);
        Console.Read();
    }
  }
}
```

---

juegoLoteria(servidor)

```
using System.Net;
using System.Net.Sockets;
```

```
using System.Text;

namespace Servidor
{
    public class Program
    {
        private readonly TcpListener listener;
        private bool listening;
        private readonly Random rdm = new Random();

        public static int Main(string[] args)
        {
            int port = 13000;
            IPAddress localAddr = IPAddress.Parse("127.0.0.1");
            Program servidor = new Program(localAddr, port);

            servidor.juegoLoteria();

            return 0;
        }
        public Program(IPAddress address, int port)
        {
            listener = new TcpListener(address, port);
            juegoLoteria();
        }

        public bool Listening => listening;

        private readonly int nmax = 101;
        private readonly int nmin = 1;

        private void juegoLoteria()
        {
            int numeroSecreto = rdm.Next(nmin, nmax);
            Console.WriteLine("El numero aleatorio es: {0}", numeroSecreto);

            listener.Start();
            listening = true;
            Console.WriteLine("Socket lister creado.");
            bool finPartida = false;
            int idGanador = 0;
            int idPerdedor = 0;
            //Console.ReadKey();
            object o = new object();
            lock (o)
            {
                try
                {
                    while (true)
                    {
                        Task.Run(async () =>
                        {
                            int id = (int)Task.CurrentId;
                            string mensaje = "Identificador de cliente: " + id + "\nIntenta adivinar mi numero:";

                            TcpClient cliente = await listener.AcceptTcpClientAsync();
                            using (NetworkStream? networkStream = cliente.GetStream())
                            {
                                Console.WriteLine("Conexion con cliente establecida.");
```

```
    byte[]? buffer = new byte[4096];
    Console.WriteLine("Buffer de entrada y salida creados.\nIdentificadorCliente: {0}", id);
    while (true)
    {
        int byteCount = await networkStream.ReadAsync(buffer, 0, buffer.Length);
        string request = Encoding.UTF8.GetString(buffer, 0, byteCount);

        if (finPartida == true && id != idGanador && id != idPerdedor)

        {
            idPerdedor = id;
            Console.WriteLine(request);
            request = "El ganador es: " + idGanador.ToString();
            Console.WriteLine("0\nGANADOR\nIdentificador: {0}\nCliente: 0\nHas acertado!!Zorionak!", idGanador);
        }
        else
        {


            if (request.Equals("cliente"))
            {
                request = id.ToString();

            }
            else if (int.Parse(request) > numeroSecreto)
            {
                Console.WriteLine(request);
                request = "El numero es menor.";

            }
            else if (int.Parse(request) < numeroSecreto)
            {
                Console.WriteLine(request);
                request = "El numero es mayor.";

            }
            else if (int.Parse(request) == numeroSecreto)
            {
                Console.WriteLine(request);
                request = "Has acertado!!Zorionak!";
                Console.WriteLine(request);
                finPartida = true;
                idGanador = id;

            }
            else
            {
                return;
            }

        }

        byte[] ServerResponseBytes = Encoding.UTF8.GetBytes(request);
        await networkStream.WriteAsync(ServerResponseBytes, 0, ServerResponseBytes.Length);

    }
}
```

```
                    });
                }
            }


            catch (Exception e)
            {
                Console.WriteLine("Exception: {0}", e);
            }
            finally
            {
                listening = false;
            }
        }
    }
}
```

---

fotoClienteServidor(Cliente)

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;

namespace PSP03_Socket_TCP
{
    internal class Cliente
    {
        public static int Main(String[] args)
        {

            Cliente cliente = new Cliente();
            cliente.FuncionServidor();

            Console.WriteLine("Pulse intro para continuar");
            Console.ReadLine();

            return 0;
        }
        private void FuncionServidor()
        {
            Socket sender = null;
            try
            {
                int port = 12000;
                string data = null;


                IPHostEntry ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());
```

```
IPAddress ipAddress = ipHostInfo.AddressList[1];

sender = new Socket(ipAddress.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
Console.WriteLine("Programa cliente iniciando.\n");

//Conexión de socket al servidor
IPEndPoint iPEndPoint = new IPEndPoint(ipAddress.Address, port); //Indicamos IP de servidor y puerto del servidor
sender.Connect(iPEndPoint); //Se establece la conexión
Console.WriteLine("Socket conectado a servidor {0}\n", sender.RemoteEndPoint.ToString()); //Mostramos por pantalla que todo ha ido correcto



//Recepción de información
Console.WriteLine("Cliente preparado para la transferencia de datos con servidor.\n");

//Menu
bool salir = false;

while (!salir)
{

    try
    {
        Console.WriteLine("Elija una de las siguientes imágenes para su descarga:\n" +
            "**************************************************\n");
        Console.WriteLine("1. FotoMonte");
        Console.WriteLine("2. FotoPlaya");
        Console.WriteLine("3. FotoCiudad");
        Console.WriteLine("4. Salir");
        Console.WriteLine("Elige una de las opciones:");

        data = string.Empty;
        data = Console.ReadLine();

        int opcion = Convert.ToInt32(data);

        switch (opcion)
        {
            case 1:
                DescargaFichero("FotoMonte", data, sender);
                break;

            case 2:
                DescargaFichero("FotoPlaya", data, sender);
                break;

            case 3:
                DescargaFichero("FotoCiudad", data, sender);
                break;
            case 4:
                Console.WriteLine("Has elegido salir de la aplicación");
                salir = true;
```

```
                    break;
                default:
                    Console.WriteLine("Elige una opcion entre 1 y 4");
                    break;
            }

        }
        catch (FormatException e)
        {
            Console.WriteLine(e.Message);
        }
    }
    Console.ReadLine();
    sender.Close();

}
catch (Exception ex)
{
    Console.WriteLine(ex.ToString());
}
finally
{
    //Cerramos el socket
    sender.Close();

}
}


//El siguiente método tiene como objetivo realizar el descarga del fichero desde el servidor.
//@nombreFoto: Se recoge el nombre de la foto.
//@sender: Se recoge el objeto socket creado para la comunicación con el servidor.
//@opcion: Recoge la opción del menú en una variable entera.
private void DescargaFichero(string nombreFoto, string opcion, Socket sender)
{
    byte[] bytes = new Byte[1024];
    byte[] tamanob = new Byte[16];
    string ruta =string.Empty;
    string data = string.Empty;



    //Envío de opción
    Console.WriteLine("Has elegido descargar {0}.", nombreFoto);

    Console.WriteLine("Comienza la descarga...espere unos segundos");
    bytes = Encoding.ASCII.GetBytes(opcion);
    sender.Send(bytes);

    //Recoger tamaño fichero
    int bytesRec = sender.Receive(tamanob, sizeof(long), SocketFlags.None);
    data = Encoding.ASCII.GetString(tamanob, 0, bytesRec);
```

```
            var tamano = Convert.ToInt32(data);


            //Especificar ruta donde se va a guardar la imagen
            ruta = String.Empty;
            ruta = @"../../../argazkiak/";
            ruta = ruta + nombreFoto + ".jpg";

            try
            {
                //Escritura de fichero en tamaño de bloques
                int bytesReadTotal = 0;
                using (FileStream fs = File.OpenWrite(ruta))
                {

                    while (bytesReadTotal < tamano)
                    {
                        //Recibe fichero en bloques de 1024.
                        int bytesRead = sender.Receive(bytes, bytes.Length, SocketFlags.None);
                        //Guarda bloque de fichero
                        fs.Write(bytes, 0, bytesRead);
                        //Aumenta contados hasta que el total recibido sea igual que el tamaño total del fichero.
                        bytesReadTotal += bytesRead;

                    }

                };
            }catch (FileNotFoundException e)
            {
                Console.WriteLine("Fichero no encontrado: {0}", e.Message);
            }


    //Muestra por consola la descarga del fichero
    Console.WriteLine("Finalizada la descarga, fichero {0} guardado en {1}", nombreFoto, Path.GetFullPath(ruta));
            nombreFoto = string.Empty;

        }
    }
}
```

---

fotoClienteServidor(Servidor)

```
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;
```

```
namespace PSP03_Socket_TCP
{
    internal class Servidor
    {
        public static int Main(String[] args)
        {

            Servidor servidor = new Servidor();
            servidor.FuncionCliente();

            Console.WriteLine("Pulse intro para continuar");
            Console.ReadLine();

            return 0;
        }
        private void FuncionCliente()
        {
            Socket listener = null;
            Socket handler = null;


            try
            {
                //declaramos el puerto
                int port = 12000;
                string data = null;

                //buffer
                byte[] opcion = new Byte[1028];

                //Recogemos la IP del servidor
                IPHostEntry ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());
                IPAddress ipAddress = ipHostInfo.AddressList[1];

                //Creación del socket listener para recepcionar las peticiones del cliente
                listener = new Socket(ipAddress.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
                Console.WriteLine("Programa servidor iniciando.");

                //Asociamos el socket al puerto e ip del servidor
                IPEndPoint iPEndPoint = new IPEndPoint(ipAddress.Address, port);
                listener.Bind(iPEndPoint);

                //Quedamos a la escucha de un máximo de peticiones de cliente de 10 (en este caso sólo se trabajará con 1 cliente).
                listener.Listen(10);

                //Se establece la conexión con el cliente y abre un segundo socket para la comunicación
                handler = listener.Accept(); //Bloqueante.
                Console.WriteLine("Aceptada la conexión con el  cliente.");

                bool salir = false;
```

```
        //Recepción de información
        while (!salir)
        {
            int bytesRec = handler.Receive(opcion); //El cliente envía la opción a elegir
            data += Encoding.ASCII.GetString(opcion, 0, bytesRec);

            //En base al menú transmitimos una imagen u otra.
            switch (data)
            {
                case "1":
                    EnviarFichero("FotoMonte", handler);
                    data = String.Empty; //Reinicialmos el dato de selección de opción
                    break;

                case "2":
                    EnviarFichero("FotoPlaya", handler);
                    data = String.Empty;
                    break;

                case "3":
                    EnviarFichero("FotoCiudad", handler);
                    data = String.Empty;
                    break;

                case "4":
                    data = String.Empty;
                    salir = true;
                    break;
                default:
                    Console.WriteLine("Se cierra por error.");
                    salir = true;
                    data = String.Empty;
                    break;
            }

        }
        Console.ReadLine();
        handler.Close();
        listener.Close();

    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
    finally
    {
        handler.Close();
        listener.Close();
    }

}
```

```
//El siguiente método tiene como objetivo realizar el envío del fichero a cliente.
//@nombreFoto: Se recoge el nombre de la foto.
//@handler: Se recoge el objeto socket creado para la comunicación con el cliente.
private void EnviarFichero(string nombreFoto, Socket handler)
{
    FileInfo fileinfo = null;
    long tamano = long.MinValue;

    string ruta = String.Empty;
    //Creamos el buffer para el envío y recepción de información
    byte[] bytes = new Byte[1028];


    ruta = @"../../../argazkiakhartu/" + nombreFoto + ".jpg";
    Console.WriteLine("Foto enviada con ruta:{0}", ruta);

    //Recogemos tamaño del fichero (El fichero que se genere en destino sea igual que el origen)
    fileinfo = new FileInfo(ruta);
    tamano = fileinfo.Length;

    bytes = Encoding.ASCII.GetBytes(Convert.ToString(tamano));

    //Se envía tamaño de fichero
    handler.Send(bytes, bytes.Length, SocketFlags.None);

    //Comienza el envío de la imagen
    try
    {
        int bytesReadTotal = 0; //init de variable para el control del tamaño del fichero

        //Lectura de fichero en bloques de 1024bytes y transmisión de cada bloque
        using (FileStream fs = File.OpenRead(ruta))
        {
            while (bytesReadTotal < tamano)
            {
                //Lectura de fichero en bloques de 1024. El bloque se guarda en el buffer denominado bytes
                int bytesRead = fs.Read(bytes, 0, bytes.Length);

                //Envío de bloque leido (bytes)
                handler.Send(bytes, bytesRead, SocketFlags.None);

                //Aumenta el contador del tamaño del fichero leido, hasta que alcance el total
                bytesReadTotal += bytesRead;
            }

        };
    }catch (FileNotFoundException e)
    {
        Console.WriteLine("Fichero no encontrado: {0}", e.Message);
    }
```

```
        }
    }
}
```

---

servidorFTP(Cliente)

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Text;
using System.Threading.Tasks;
using System.Net.Mail;

namespace clienteFTP
{
    public class manipuladorCliente
    {

        public void descargaFicheros(string user, string pwd, string url, string seleccionDescarga, string pathDescarga)
        {
            FtpWebRequest request = (FtpWebRequest)WebRequest.Create("ftp://" + user + ":" + pwd + "@" + url + "/" + seleccionDescarga);
            request.Credentials = new NetworkCredential(user, pwd);

            FtpWebResponse response = (FtpWebResponse)request.GetResponse();

            Stream responseStream = response.GetResponseStream();
            StreamReader reader = new StreamReader(responseStream);
            string arreglado = pathDescarga;

            using (FileStream writer = new FileStream(pathDescarga + "\\" + seleccionDescarga, FileMode.Create))
            {

                long length = response.ContentLength;
                int bufferSize = 32768;
                int readCount;
                byte[] buffer = new byte[32768];

                readCount = responseStream.Read(buffer, 0, bufferSize);
                while (readCount > 0)
                {
                    writer.Write(buffer, 0, readCount);
                    readCount = responseStream.Read(buffer, 0, bufferSize);
```

```
        }
    }

    reader.Close();
    response.Close();
}

public void subirFicheros(string user, string pwd, string url, string path, string nombreFicheroSubida)
{
    string nombreFichero = Path.GetFileName(path);
    string urlSubida;
    if (nombreFicheroSubida.Length > 0)
        urlSubida = "ftp://" + url + "//" + nombreFicheroSubida;
    else
        urlSubida = "ftp://" + url + "//" + nombreFichero;

    var request = (FtpWebRequest)WebRequest.Create(urlSubida);

    request.Method = WebRequestMethods.Ftp.UploadFile;
    request.Credentials = new NetworkCredential(user, pwd);
    request.UsePassive = true;
    request.UseBinary = true;
    request.KeepAlive = false;

    using (var fileStream = File.OpenRead(path))
    {
        using (var requestStream = request.GetRequestStream())
        {
            fileStream.CopyTo(requestStream);
            requestStream.Close();
        }
    }

    var response = (FtpWebResponse)request.GetResponse();
    response.Close();
}


public void enviarEmailSubida(string emailSubida, string seleccionSubida, string url)
{

    string nombre = string.Empty;
    for (int i = 0; i < emailSubida.Length; i++)
    {
        if (emailSubida[i] == '@')
            break;

        nombre += emailSubida[i];
    }
```

```
    MailAddress origen = new MailAddress("pepitopiscinascooperativa@gmail.com", "Pepito Piscinas Coop.");

    MailAddress destino = new MailAddress(emailSubida, nombre);

    SmtpClient smtp = new SmtpClient
    {
        Host = "smtp.gmail.com",
        Port = 587,
        Credentials = new NetworkCredential(origen.Address, "PSP04JSR"),
        EnableSsl = true
    };



    using (MailMessage mensaje = new MailMessage(origen, destino)
    {
        Subject = "Confirmación Subida Fichero",
        Body = "Su fichero " + seleccionSubida + " se ha subido a " + url,
    })

        try
        {
            smtp.Send(mensaje);

        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.ToString());
        }

}

public void enviarEmailDescarga(string emailDescarga, string seleccionDescarga, string url)
{

    string nombre = string.Empty;
    for (int i = 0; i < emailDescarga.Length; i++)
    {
        if (emailDescarga[i] == '@')
            break;

        nombre += emailDescarga[i];
    }

    MailAddress origen = new MailAddress("pepitopiscinascooperativa@gmail.com", "Pepito Piscinas Coop.");

    MailAddress destino = new MailAddress(emailDescarga, nombre);

    SmtpClient smtp = new SmtpClient
```

```
            {
                Host = "smtp.gmail.com",
                Port = 587,
                Credentials = new NetworkCredential(origen.Address, "PSP04JSR"),
                EnableSsl = true
            };



            using (MailMessage mensaje = new MailMessage(origen, destino)
            {
                Subject = "Confirmación Descarga Fichero",
                Body = "Su fichero " + seleccionDescarga + " se ha descargado desde " + url,
            })

                try
                {
                    smtp.Send(mensaje);

                }
                catch (Exception ex)
                {
                    Console.WriteLine(ex.ToString());
                }



        }

    }
}
```

---

servidorFTP(servidor)

```
public string[] GetListaFicheros()
        {
            string[] downloadFiles;

            StringBuilder result = new StringBuilder();
            bool extension = false;
            WebResponse response = null;
            StreamReader reader = null;
            try
            {
                FtpWebRequest reqFTP;
                reqFTP = (FtpWebRequest)FtpWebRequest.Create(new Uri("ftp://" + url));
```

```
            reqFTP.UseBinary = true;
            reqFTP.Credentials = new NetworkCredential(user, pwd);
            if (listarDetalles == true)
                reqFTP.Method = WebRequestMethods.Ftp.ListDirectoryDetails;
            else
                reqFTP.Method = WebRequestMethods.Ftp.ListDirectory;
            reqFTP.Proxy = null;
            reqFTP.KeepAlive = false;
            reqFTP.UsePassive = false;
            response = reqFTP.GetResponse();
            reader = new StreamReader(response.GetResponseStream());
            string line = reader.ReadLine();
            while (true)
            {

                if (line == null)
                {
                    break;
                }
                if (listarDespegable)
                {
                    if (line == "." || line == "..")
                    {
                        line = reader.ReadLine();
                        continue;
                    }
                }


                for (int i = line.Length - 1; i >= 0; i--)
                {
                    if (line[i] == '.')
                    {
                        extension = true;
                        break;
                    }
                }

                if (extension || (!extension && !listarDespegable))
                {
                    result.Append(line);
                    result.Append("\n");
                    extension = false;
                }

                line = reader.ReadLine();

            }

            result.Remove(result.ToString().LastIndexOf('\n'), 1);
            nombreFicheros = result.ToString().Split('\n');
            return nombreFicheros;

        }

}
```

```
private void botonConectar_Click(object sender, EventArgs e)
        {
            /*
            user = "asfasf392";
            pwd = "w23w4SHYl";
            url = "ftps4.us.freehostia.com";
            */
            listarDetalles = false;

            var conexion = new FtpClient(url, user, pwd);
            try
            {
                using (conexion)
                {
                    conexion.Connect();
                    GetListaFicheros();
                    string delim = "\r\n";
                    string nombreficheros = string.Empty;
                    nombreficheros = nombreFicheros.Aggregate((prev, current) => prev + delim + current);

                    listadoFicheros.Text = nombreficheros;
                    cajaDespegable.Items.Clear();
                    cajaDespegable.Items.AddRange(nombreFicheros);
                }
            }
            catch (Exception ex)
            {
                if (ex is FtpAuthenticationException || ex is SocketException)
                    listadoFicheros.Text = ex.Message;
            }
        }
```

```
    private void botonListarContenido_Click(object sender, EventArgs e)
        {
            listarDetalles = true;

            var conexion = new FtpClient(url, user, pwd);
            try
            {
                using (conexion)
                {
                    conexion.Connect();
                    GetListaFicheros();
                    string delim = "\r\n";
                    string nombreficheros = string.Empty;
                    nombreficheros = nombreFicheros.Aggregate((prev, current) => prev + delim + current);

                    listadoFicheros.Text = nombreficheros;
                    listarDetalles = false;
                    GetListaFicheros();
                    cajaDespegable.Items.Clear();
                    cajaDespegable.Items.AddRange(nombreFicheros);
                }
```

```
        }
        catch (Exception ex)
        {
            if (ex is FtpAuthenticationException || ex is SocketException)
                listadoFicheros.Text = ex.Message;
        }
    }
```

```
    private void cajaDespegable_abrirDespegable(object sender, EventArgs e)
    {
        var conexion = new FtpClient(url, user, pwd);
        try
        {
            using (conexion)
            {
                conexion.Connect();
                listarDetalles = false;
                listarDespegable = true;
                GetListaFicheros();
                cajaDespegable.Items.Clear();
                cajaDespegable.Items.AddRange(nombreFicheros);
                listarDespegable = false;
            }
        }
        catch (Exception ex)
        {
            if (ex is FtpAuthenticationException || ex is SocketException)
                listadoFicheros.Text = ex.Message;
        }

    }
```

```
    private void botonUbicacionDescarga_Click(object sender, EventArgs e)
    {
        CommonOpenFileDialog dialog = new CommonOpenFileDialog();
        dialog.InitialDirectory = "C:\\";
        dialog.IsFolderPicker = true;
        if (dialog.ShowDialog() == CommonFileDialogResult.Ok)
        {
            pathDescarga = dialog.FileName.ToString();
            ubicacionDescarga.Text = pathDescarga;
        }
    }
```

```
    private void botonDescargarFichero_Click(object sender, EventArgs e)
    {

        manipulador.descargaFicheros(user, pwd, url, seleccionDescarga, pathDescarga);

        if (textBoxEmailDescarga.Text != "" && checkBoxConfirmacionDescarga.Checked)
        {
            manipulador.enviarEmailDescarga(textBoxEmailDescarga.Text, seleccionDescarga, url);
            textBoxEmailDescarga.Text = string.Empty;
        }
```

```
            ubicacionDescarga.Text = string.Empty;
        }
```

```
        private void botonSeleccionarFicheroSubida_Click(object sender, EventArgs e)
        {
            OpenFileDialog ofd = new OpenFileDialog();
            DialogResult result = ofd.ShowDialog();
            if (result == DialogResult.OK)
            {
                ficheroSubida = ofd.FileName;

                seleccionSubida = File.ReadAllText(ficheroSubida);
                ubicacionArchivoSubida.Text = ficheroSubida;

            }


        }
```

---

passEncriptadas

```
namespace comparadorFicheros
{
    public class comparadorFicheros
    {

        public bool FileCompare(string file1, string file2)
        {
            try
            {
                int file1byte;
                int file2byte;
                FileStream fs1;
                FileStream fs2;

                if (file1 == file2)
                {
                    return true;
                }

                fs1 = new FileStream(file1, FileMode.Open);
                fs2 = new FileStream(file2, FileMode.Open);


                if (fs1.Length != fs2.Length)
                {
                    fs1.Close();
                    fs2.Close();
```

```
                return false;
            }

            do
            {
                file1byte = fs1.ReadByte();
                file2byte = fs2.ReadByte();
            }
            while ((file1byte == file2byte) && (file1byte != -1));

            fs1.Close();
            fs2.Close();

            return ((file1byte - file2byte) == 0);
        }
        catch (Exception ex) { MessageBox.Show("Archivo innaccesible"); Console.WriteLine(ex.ToString()); return false; }
    }

}
}
```

```
namespace Correos
{
    public class envioCorreos
    {

        public void envioEmail(string usuario, string correo)
        {
            try
            {
                MailMessage mensaje = new MailMessage("pepitopiscinascooperativa@gmail.com", correo, "Clave Privada", "Clave de acceso a contraseñas en el gestor de password.");
                Attachment data = new Attachment(@"..\..\privatekeys\" + usuario + "_private.xml", MediaTypeNames.Application.Octet);
                mensaje.Attachments.Add(data);
                SmtpClient smtp = new SmtpClient
                {
                    Host = "smtp.gmail.com",
                    Port = 587,
                    Credentials = new NetworkCredential("pepitopiscinascooperativa@gmail.com", "PSP04JSR"),
                    EnableSsl = true,


                };


                smtp.Send(mensaje);

            }
            catch (Exception ex)
            {
                MessageBox.Show("No se ha podido mandar el correo."); Console.WriteLine(ex.ToString());
            }
```

```
        }
    }
}
```

```
private byte[] bytextoCifrado = new byte[2048 * 2];

        private string pathPublicKey = "..\\..\\publickeys\\";
        private string pathPrivateKey = "..\\..\\privatekeys\\";
        private string pathBBDD = "..\\..\\bbdd\\";
        List<string> Passwords = new List<string>();
        public Form1()
```

```
        private void generarClaves(string pathPublicKey, string pathPrivateKey)
        {
            try
            {
                using (var rsa = new RSACryptoServiceProvider(2048 * 2))
                {

                    rsa.PersistKeyInCsp = false;

                    if (File.Exists(pathPublicKey))
                        File.Delete(pathPublicKey);
                    if (File.Exists(pathPrivateKey))
                        File.Delete(pathPrivateKey);


                    string publicKey = rsa.ToXmlString(false);

                    File.WriteAllText(pathPublicKey, publicKey);


                    string privateKey = rsa.ToXmlString(true);
                    File.WriteAllText(pathPrivateKey, privateKey);

                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("No se han podido generar las claves.");
                Console.WriteLine(ex.ToString());
            }
        }
```

```
        public static byte[] encriptar(string publicKF, byte[] textoPlano)
        {
            try
```

```
            {
                byte[] encriptado;

                using (var rsa = new RSACryptoServiceProvider(2048 * 2))
                {
                    rsa.PersistKeyInCsp = false;

                    string publicKey = File.ReadAllText(publicKF);

                    rsa.FromXmlString(publicKey);


                    encriptado = rsa.Encrypt(textoPlano, true);

                }


                return encriptado;
            }
            catch (Exception ex)
            {
                MessageBox.Show("No se ha podido encriptar.");
                Console.WriteLine(ex.ToString());
                return Encoding.UTF8.GetBytes(ex.ToString());
            }

        }
```

```
public static byte[] Desencriptar(string privateKF, byte[] textoEncriptado)
        {
            try
            {
                byte[] desencriptado;
                using (var rsa = new RSACryptoServiceProvider(2048 * 2))
                {
                    rsa.PersistKeyInCsp = false;


                    string privateKey = File.ReadAllText(privateKF);


                    rsa.FromXmlString(privateKey);


                    desencriptado = rsa.Decrypt(textoEncriptado, true);

                }
                return (desencriptado);
            }
            catch (Exception ex)
            {
                MessageBox.Show("No se ha podido desencriptar.");
                Console.WriteLine(ex.ToString());
                return Encoding.UTF8.GetBytes(ex.ToString());
            }
        }
```

```
        private void boton_Guardar_Click(object sender, EventArgs e)
        {
            try
            {
                if (tb_RegistrarDescripcion.Text.Length > 0 && tb_RegistrarPassword.Text.Length > 0)
                {
                    char c;
                    bool mayus = false;
                    bool minus = false;
                    bool numero = false;
                    bool longitud = false;
                    bool caracter = false;
                    string caracteres = "!@#&()-[{}:',?/*~$^+=<>";

                    if (tb_RegistrarPassword.Text.Length > 7 && tb_RegistrarPassword.Text.Length < 21)
                    {
                        longitud = true;
                        for (int i = 0; i < tb_RegistrarPassword.Text.Length; ++i)
                        {
                            c = tb_RegistrarPassword.Text[i];
                            if (c >= 'A' && c <= 'Z') { mayus = true; }
                            else if (c >= 'a' && c <= 'z') { minus = true; }
                            else if (c >= '0' && c <= '9') { numero = true; }
                            else if (caracteres.Contains(c.ToString())) { caracter = true; }
                        }
                    }
                    if (!longitud || !mayus || !minus || !numero || !caracter)
                    {

                        MessageBox.Show("La contraseña al menos tiene que contener\n1 mayúscula\n1 minúscula\n1 número\n8-10 caracteres de longitud\n1 caracter: !@#&()-
[{}:',?/*~$^+=<>");
                    }
                    else
                    {
                        byte[] textoAnteriorBytes = File.ReadAllBytes(pathBBDD + tb_UsuarioRegistrado.Text + ".txt");
                        byte[] textoAnteriorBytesDesencriptado = Desencriptar(pathPrivateKey + tb_UsuarioRegistrado.Text + "_private.xml", textoAnteriorBytes);
                        string textoPlanoString = tb_RegistrarDescripcion.Text + "." + tb_RegistrarPassword.Text + ";";
                        byte[] textoPlanoByte = Encoding.UTF8.GetBytes(textoPlanoString);
                        byte[] textoCombinado = Combine(textoAnteriorBytesDesencriptado, textoPlanoByte);
                        byte[] textoCombinadoEncriptado = encriptar(pathPublicKey + tb_UsuarioRegistrado.Text + "_public.xml", textoCombinado);
                        File.WriteAllBytes(pathBBDD + tb_UsuarioRegistrado.Text + ".txt", textoCombinadoEncriptado);
                    }
                }
            }
            catch (Exception ex) { MessageBox.Show("Contraseña no guardada."); Console.WriteLine(ex.ToString()); }


        }
```

```
        private void check_VisualizarPass_CheckedChanged(object sender, EventArgs e)
        {
            try
            {
                if (check_VisualizarPass.Checked)
                {
                    gb_Visualizar.Enabled = true;
                    cb_VisualizarDescripcion.Items.Clear();
                    cb_VisualizarDescripcion.Text = "";
```

```
                byte[] textoDescifrar = File.ReadAllBytes(pathBBDD + tb_UsuarioRegistrado.Text + ".txt");

                byte[] desencriptado = Desencriptar(pathPrivateKey + tb_UsuarioRegistrado.Text + "_private.xml", textoDescifrar);
                string textoCompleto = Encoding.UTF8.GetString(desencriptado);

                string programa = string.Empty;
                string password = string.Empty;
                bool psw = false;
                for (int i = 0; i < textoCompleto.Length; ++i)
                {
                    if (textoCompleto[i] == '.') { psw = true; cb_VisualizarDescripcion.Items.Add(programa); programa = string.Empty; continue; }
                    else if (textoCompleto[i] == ';') { psw = false; Passwords.Add(password); password = string.Empty; continue; }
                    if (!psw)
                        programa += textoCompleto[i];
                    else
                        password += textoCompleto[i];

                }


            }
            else { gb_Visualizar.Enabled = false; }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Ha habido un error.");
            Console.WriteLine(ex.ToString());
        }
    }
```

```
private void check_RegistrarPass_CheckedChanged(object sender, EventArgs e)
    {
        try
        {
            if (check_RegistrarPass.Checked) { gb_Registrar.Enabled = true; }
            else { gb_Registrar.Enabled = false; }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Ha habido un error.");
            Console.WriteLine(ex.ToString());
        }
    }

    private void check_BorrarPass_CheckedChanged(object sender, EventArgs e)
    {
        try
        {
            if (check_BorrarPass.Checked)
            {

                gb_Borrar.Enabled = true;
                cb_BorrarDescripcion.Items.Clear();
```

```
                cb_BorrarDescripcion.Text = "";
                byte[] textoDescifrar = File.ReadAllBytes(pathBBDD + tb_UsuarioRegistrado.Text + ".txt");

                byte[] desencriptado = Desencriptar(pathPrivateKey + tb_UsuarioRegistrado.Text + "_private.xml", textoDescifrar);
                string textoCompleto = Encoding.UTF8.GetString(desencriptado);

                string programa = string.Empty;
                string password = string.Empty;
                bool psw = false;
                for (int i = 0; i < textoCompleto.Length; ++i)
                {
                    if (textoCompleto[i] == '.') { psw = true; cb_BorrarDescripcion.Items.Add(programa); programa = string.Empty; continue; }
                    else if (textoCompleto[i] == ';') { psw = false; Passwords.Add(password); password = string.Empty; continue; }
                    if (!psw)
                        programa += textoCompleto[i];
                    else
                        password += textoCompleto[i];

                }
            }
            else { gb_Borrar.Enabled = false; }
        }
        catch (Exception ex)
        {
            MessageBox.Show("Ha habido un error.");
            Console.WriteLine(ex.ToString());
        }
    }
```

```
private void boton_RegistrarUsuario_Click(object sender, EventArgs e)
    {
        try
        {
            string nombreUsuario = tb_UsuarioRegistrado.Text;
            string curFile = @"..\..\bbdd\" + nombreUsuario + ".txt";
            if (File.Exists(curFile))
            {
                check_BorrarPass.Enabled = true;
                check_RegistrarPass.Enabled = true;
                check_VisualizarPass.Enabled = true;
            }
            else
            {
                MessageBox.Show("El usuario no existe, debes registrarlo.");
                gb_RegistroUsuario.Enabled = true;
            }

        }
        catch (Exception ex)
        {
            MessageBox.Show("Ha habido un error.");
            Console.WriteLine(ex.ToString());
        }

    }
```

```
        private void boton_RegistroAceptar_Click(object sender, EventArgs e)
        {
            try
            {
                bool emailCorrecto = false;
                char c;
                bool arroba = false;
                for (int i = 0; i < tb_Email.Text.Length; ++i)
                {
                    if (i == tb_Email.Text.Length - 1 && arroba == true)
                        emailCorrecto = true;

                    c = tb_Email.Text[i];

                    if ((c > 44 && c < 58) || (c > 64 && c < 91) || (c > 96 && c < 123))
                        continue;

                    else if (c == '@' && arroba == false)
                        arroba = true;

                    else
                        break;

                }
                if (radio_RegistrarYes.Checked && emailCorrecto)
                {
                    generarClaves(pathPublicKey + tb_UsuarioRegistrado.Text + "_public.xml", pathPrivateKey + tb_UsuarioRegistrado.Text + "_private.xml");
                    bytextoCifrado = encriptar(pathPublicKey + tb_UsuarioRegistrado.Text + "_public.xml", Encoding.UTF8.GetBytes(""));


                    File.WriteAllBytes(pathBBDD + tb_UsuarioRegistrado.Text + ".txt", bytextoCifrado);

                    gb_RegistroUsuario.Enabled = false;
                    int tiempo = 0;
                    while (!File.Exists(@"..\..\privatekeys\" + tb_UsuarioRegistrado.Text + "_private.xml"))
                    {
                        Thread.Sleep(200);
                        tiempo++;
                        if (tiempo > 1500)
                        {
                            MessageBox.Show("Tiempo excesivo esperando a la creación del archivo, inténtalo otra vez");
                            return;
                        }
                    }
                    Correos.envioCorreos enviarCorreo = new Correos.envioCorreos();

                    enviarCorreo.envioEmail(tb_UsuarioRegistrado.Text.ToString(), tb_Email.Text.ToString());

                    MessageBox.Show("Registro Creado, puedes encontrar tu fichero en ..\\..\\privatekeys\\" + tb_UsuarioRegistrado.Text + "_private.xml");
                    Application.Restart();

                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Ha habido un error.");
                Console.WriteLine(ex.ToString());
            }
```

```
        }
```

```
public static byte[] Combine(byte[] first, byte[] second)
        {
            try
            {
                byte[] ret = new byte[first.Length + second.Length];
                Buffer.BlockCopy(first, 0, ret, 0, first.Length);
                Buffer.BlockCopy(second, 0, ret, first.Length, second.Length);
                return ret;
            }
            catch (Exception ex)
            {
                MessageBox.Show("Ha habido un error.");
                Console.WriteLine(ex.ToString());
                return Encoding.UTF8.GetBytes(ex.ToString());
            }
        }

        private void boton_Fichero_Click(object sender, EventArgs e)
        {
            try
            {
                using (OpenFileDialog openFileDialog = new OpenFileDialog())
                {
                    openFileDialog.InitialDirectory = "c:\\";
                    openFileDialog.Filter = "xml files (*.xml)|*.xml|All files (*.*)|*.*";
                    openFileDialog.FilterIndex = 1;
                    openFileDialog.RestoreDirectory = true;

                    if (openFileDialog.ShowDialog() == DialogResult.OK)
                    {
                        label_UbicacionFichero.Text = openFileDialog.FileName.ToString();
                        comparadorFicheros cf = new comparadorFicheros();


                        if (cf.FileCompare(pathPrivateKey + tb_UsuarioRegistrado.Text + "_private.xml", openFileDialog.FileName) && cb_VisualizarDescripcion.SelectedIndex >= 0)
                        {
                            tb_VisualizarPassword.Text = Passwords[cb_VisualizarDescripcion.SelectedIndex];

                        }
                        else
                        {
                            MessageBox.Show("Archivo de claves incorrecto.");
                        }

                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show("Ha habido un error.");
                Console.WriteLine(ex.ToString());
```

```
        }

    }
```

```
    private void boton_Borrar_Click(object sender, EventArgs e)
    {
        try
        {
            byte[] textoDescifrar = File.ReadAllBytes(pathBBDD + tb_UsuarioRegistrado.Text + ".txt");

            byte[] desencriptado = Desencriptar(pathPrivateKey + tb_UsuarioRegistrado.Text + "_private.xml", textoDescifrar);
            string textoCompleto = Encoding.UTF8.GetString(desencriptado);
            int contador = 0;
            int seleccionado = cb_BorrarDescripcion.SelectedIndex;
            for (int i = 0; i < textoCompleto.Length; ++i)
            {
                if (contador == seleccionado)
                {
                    while (textoCompleto[i] != ';')
                    {
                        textoCompleto = textoCompleto.Remove(i, 1);
                        continue;
                    }
                    textoCompleto = textoCompleto.Remove(i, 1);
                    contador++;
                    if (contador > seleccionado || i > textoCompleto.Length) break;
                }


                if (textoCompleto[i] == ';')
                {
                    contador++;
                    continue;
                }
            }
            byte[] textoCompletoByte = Encoding.UTF8.GetBytes(textoCompleto);
            byte[] textoCompletoEncriptado = encriptar(pathPublicKey + tb_UsuarioRegistrado.Text + "_public.xml", textoCompletoByte);

            File.WriteAllBytes(pathBBDD + tb_UsuarioRegistrado.Text + ".txt", textoCompletoEncriptado);
            cb_BorrarDescripcion.Text = string.Empty;
            cb_VisualizarDescripcion.Text = string.Empty;
        }
        catch (Exception ex)
        {
            MessageBox.Show("Ha habido un error.");
            Console.WriteLine(ex.ToString());
        }

    }
```