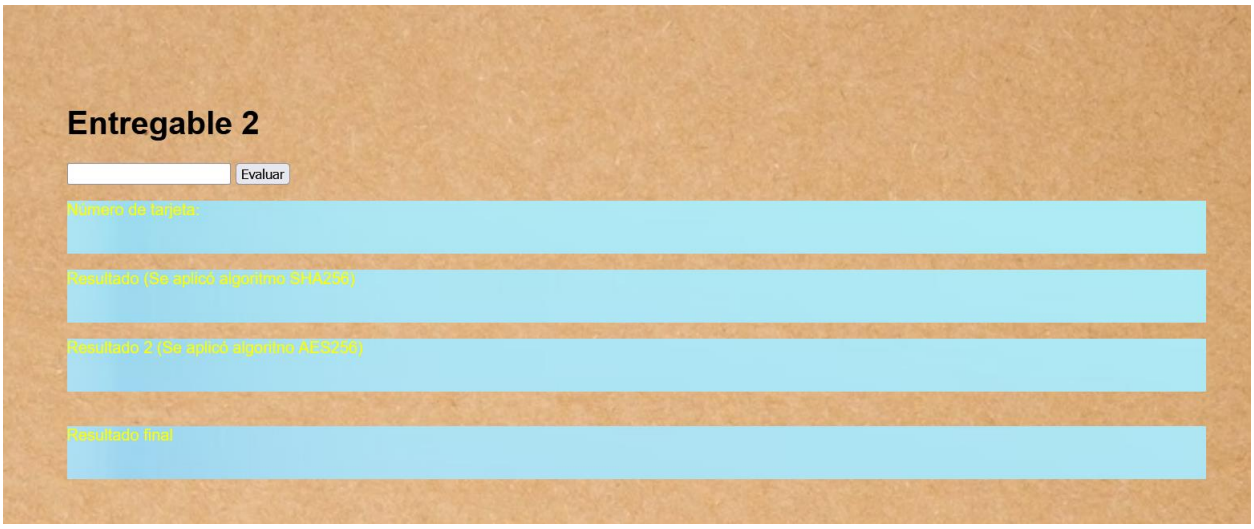
Documento de evdencias

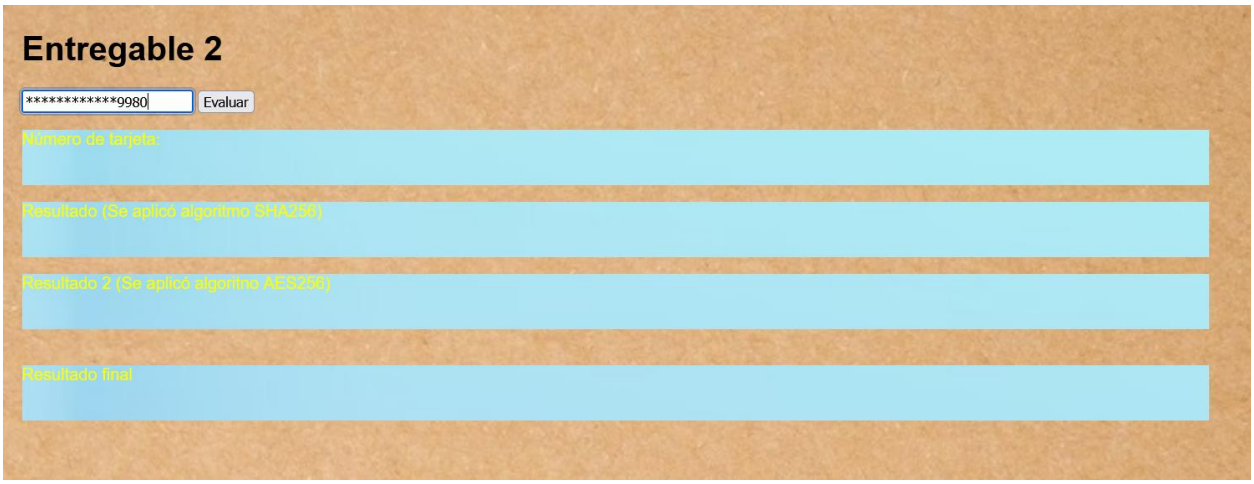
Elaborado por: Karina Guzmán Villanueva

Vista de entregable 2



Esta es la vista principal de la aplicación, a continuación se ingresará la siguiente cadena que es el numero de tarjeta

4552100120309980



Al dar clic en Evaluar muestra los resultados

Las principales funcionalidades son donde esta el botón de evaluar

```
protected void btnEnter_Click(object sender, EventArgs e)
        {
            if (txtCadena.Text.Length >= 16)
            {
                string entrada = CreditCard.Value +
txtCadena.Text.Substring(txtCadena.Text.Length - 4, 4);
                //string entrada = txtCadena.Text.Replace("-", string.Empty);
                Regex regex = new Regex(@"^\d{16,16}$");
                //Valida que sea un número de tarjeta válido
                if (regex.IsMatch(entrada))
                {
                    using (AesManaged myAes = new AesManaged())
                    {
                        string key = BitConverter.ToString(myAes.Key);
                        // Encriptar en AES256.
                        byte[] encrypted =
Controller.Encriptar.EncryptStringToBytes_Aes(entrada, myAes.Key, myAes.IV);

                        // desencriptar en AES256.
                        string roundtrip =
Controller.Encriptar.DecryptStringFromBytes_Aes(encrypted, myAes.Key, myAes.IV);

                        //Mostrando los resultados
                        lblNormal.Text = txtCadena.Text;
                        //LE aplica el algoritmo SHA512 a la cadena de entrada
                        string sSha =
BitConverter.ToString(Controller.Encriptar.HmacSha256Digest(entrada,
ConfigurationManager.AppSettings["LaKey"])).Replace("-", "").ToLower();
                        lblSHA.Text = sSha;
                        //LE aplica el algoritmo SHA512 a la cadena que previsamente se
le aplicó AES256
                        string sAES =
BitConverter.ToString(Controller.Encriptar.HmacSha256Digest(roundtrip,
ConfigurationManager.AppSettings["LaKey"])).Replace("-", "").ToLower();
                        lblAES.Text = sAES;
                        //Realiza la comparación de resultados
                        lblResultado.Text = (sSha.Equals(sAES) ? "Iguales" : "Algo falló,
no son iguales :(");
                    }
```

```
                }
                else
                {
                    txtCadena.Text = string.Empty;
                    CreditCard.Value = string.Empty;
                    //Mensaje si la tarjeta ingresada tiene un formato incorrecto
                    ClientScript.RegisterClientScriptBlock(Page.GetType(), "AlertMsg",
"<script language='javascript'>alert('Número de tarjeta incorrecto');</script>");
                }
            }
            else
            {
                txtCadena.Text = string.Empty;
                CreditCard.Value = string.Empty;
                //Mensaje si la tarjeta ingresada tiene un formato incorrecto
                ClientScript.RegisterClientScriptBlock(Page.GetType(), "AlertMsg",
"<script language='javascript'>alert('Número de tarjeta incorrecto');</script>");
            }

        }
```

Se tiene la función que aplica el algoritmo SHA512

```
  /// <summary>
        /// Función para aplicar el algoritmo SHA512
        /// </summary>
        /// <param name="message">Cadena a convertir</param>
        /// <param name="secret">Llave</param>
        /// <returns></returns>
        public static byte[] HmacSha256Digest(this string message, string secret)
        {
            ASCIIEncoding encoding = new ASCIIEncoding();
            byte[] keyBytes = encoding.GetBytes(secret);
            byte[] messageBytes = encoding.GetBytes(message);
            System.Security.Cryptography.HMACSHA256 cryptographer = new
System.Security.Cryptography.HMACSHA256(keyBytes);

            byte[] bytes = cryptographer.ComputeHash(messageBytes);
            return bytes;
        }
```

Se tiene la función que aplica el algoritmo AES256

```
        /// <summary>
        /// Función para aplicar el algoritmo AES256
        /// </summary>
        /// <param name="plainText">Texto a encriptar</param>
        /// <param name="Key"></param>
        /// <param name="IV"></param>
        /// <returns></returns>
        public static byte[] EncryptStringToBytes_Aes(string plainText, byte[] Key,
byte[] IV)
        {
            // Check arguments.
            if (plainText == null || plainText.Length <= 0)
                throw new ArgumentNullException("plainText");
```

```
            if (Key == null || Key.Length <= 0)
                throw new ArgumentNullException("Key");
            if (IV == null || IV.Length <= 0)
                throw new ArgumentNullException("IV");
            byte[] encrypted;

            // Create an AesManaged object
            // with the specified key and IV.
            using (AesManaged aesAlg = new AesManaged())
            {
                aesAlg.Key = Key;
                aesAlg.IV = IV;

                // Create an encryptor to perform the stream transform.
                ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key,
aesAlg.IV);

                // Create the streams used for encryption.
                using (MemoryStream msEncrypt = new MemoryStream())
                {
                    using (CryptoStream csEncrypt = new CryptoStream(msEncrypt,
encryptor, CryptoStreamMode.Write))
                    {
                        using (StreamWriter swEncrypt = new StreamWriter(csEncrypt))
                        {
                            //Write all data to the stream.
                            swEncrypt.Write(plainText);
                        }
                        encrypted = msEncrypt.ToArray();
                    }
                }
            }

            // Return the encrypted bytes from the memory stream.
            return encrypted;
        }
```

Se tiene la función que desencripta el algoritmo AES256

```
        /// <summary>
        /// Función para desencriptar el algoritmo AES256
        /// </summary>
        /// <param name="plainText">Texto a encriptar</param>
        /// <param name="Key"></param>
        /// <param name="IV"></param>
        /// <returns></returns>
        public static string DecryptStringFromBytes_Aes(byte[] cipherText, byte[] Key,
byte[] IV)
        {
            // Validar argumentos
            if (cipherText == null || cipherText.Length <= 0)
                throw new ArgumentNullException("cipherText");
            if (Key == null || Key.Length <= 0)
                throw new ArgumentNullException("Key");
            if (IV == null || IV.Length <= 0)
                throw new ArgumentNullException("IV");
```

```
            // string para el texto desencriptado
            string plaintext = null;

            // Create an AesManaged object
            // with the specified key and IV.
            using (AesManaged aesAlg = new AesManaged())
            {
                aesAlg.Key = Key;
                aesAlg.IV = IV;

                // Create a decryptor to perform the stream transform.
                ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key,
aesAlg.IV);

                // Create the streams used for decryption.
                using (MemoryStream msDecrypt = new MemoryStream(cipherText))
                {
                    using (CryptoStream csDecrypt = new CryptoStream(msDecrypt,
decryptor, CryptoStreamMode.Read))
                    {
                        using (StreamReader srDecrypt = new StreamReader(csDecrypt))
                        {

                            // Read the decrypted bytes from the decrypting stream
                            // and place them in a string.
                            plaintext = srDecrypt.ReadToEnd();
                        }
                    }
                }
            }

            return plaintext;
        }

    }
```