Materia:
Seguridad del software
06/Marzo/2024

Tema:
Examen RSA

Por:
Espinoza Lara Gonzalo Hazael.

Archivo: Index.cshtml

```
<script>
    function login() {
        var url = "@Url.Action("Login", "Home")";
        var email = $("#email").val();
        var password = $("#password").val();

        var data = { email: email, psw_usuario: password }

        $.ajax({
            data: data,
            url: url,
            type: 'post',
            dataType: 'json',
            success: function (response) {
                var jdata = JSON.parse(response);
                if (jdata["status"] == 200) {
                    alert("Inicio sesión correctamente")
                }
                else {
                    alert(jdata["data"]);
                }
            },
            error: function (error) {
                alert(error);
                console.log(error)
            }
        });
    }

    function registrarse() {
        var url = "@Url.Action("Registrarse", "Home")";
        var email = $("#registro-email").val();
        var password = $("#registro-password").val();

        var data = { email: email, psw_usuario: password }

        $.ajax({
            data: data,
            url: url,
            type: 'post',
            dataType: 'json',
            success: function (response) {
                if (response.status === 200) {
                    alert("Registro exitoso");
                } else {
                    alert(response.data);
                }
            },
            error: function (error) {
                alert("Error: " + error.responseText);
            }
        });
    }
</script>
```

He definido dos funciones JavaScript esenciales para un formulario de inicio de sesión y registro. La función login() se encarga de gestionar la lógica de inicio de sesión, mientras que registrarse() maneja el proceso de registro.

En el caso de login(), la función utiliza AJAX para realizar una solicitud al servidor, obteniendo la URL del controlador de inicio de sesión y los datos de correo electrónico y contraseña del formulario. En el caso de una respuesta exitosa, muestra un mensaje de alerta indicando que la sesión se inició correctamente; de lo contrario, muestra un mensaje de error proporcionado por el servidor.

La función registrarse() sigue un proceso similar pero se dirige al controlador de registro en el servidor. Recolecta los datos del formulario y realiza una solicitud AJAX. En caso de éxito, muestra una alerta de registro exitoso; de lo contrario, muestra el mensaje de error proporcionado por el servidor.

Ambas funciones utilizan jQuery para facilitar la manipulación del DOM y las solicitudes asíncronas al servidor, proporcionando una experiencia de usuario más dinámica.

```html
<div class="text-center">
    <label>Correo de usuario</label> <br />
    <input id="email" type="text" /> <br /><br />
    <label>Contraseña</label><br />
    <input id="password" type="password" />  <br /><br />
    <button onclick="login()">Iniciar Sesión</button> <br /><br />
    <button onclick="showRegistroForm()" style="background-color: r
```

Este fragmento HTML representa un formulario de inicio de sesión. Hay campos para ingresar el correo electrónico y la contraseña. Cuando se hace clic en el botón "Iniciar Sesión", se llama a la función login(). Además, hay un botón "Registrarse" que, cuando se presiona, activa la función showRegistroForm() para mostrar el formulario de registro.

```html
<div id="registro-container" style="display: none;">
    <h2>Registrarse</h2>
    <label>Correo de usuario</label> <br />
    <input id="registro-email" type="text" /> <br /><br />
    <label>Contraseña</label><br />
    <input id="registro-password" type="password" /> <br /><br />
    <button onclick="registrarse()">Registrarse</button>
</div>
```

Aquí, he creado un contenedor (div) para el formulario de registro. Inicialmente, está oculto (display: none;). Cuando se presiona el botón "Registrarse" en el formulario de inicio de sesión, se activa esta sección y se muestra el formulario de registro.

```html
<script>
    function showRegistroForm() {
        $("#registro-container").show();
        $(".text-center").hide();
    }

    function showLogin() {
        $("#registro-container").hide();
        $(".text-center").show();
    }
</script>
```

Finalmente, he definido dos funciones adicionales. showRegistroForm() se encarga de hacer visible el formulario de registro y ocultar el de inicio de sesión. showLogin() se podría usar para revertir esta acción, aunque no se utiliza directamente en el código proporcionado.

Archivo: HomeController.cs

```
private readonly ILogger<HomeController> _logger;

0 referencias
public HomeController(ILogger<HomeController> logger)
{
    _logger = logger;
}
```

En el constructor HomeController, estoy definiendo una variable privada _logger del tipo ILogger<HomeController>. Este logger se utiliza para realizar registros o logs, lo cual es útil para el seguimiento y diagnóstico de eventos en la aplicación.

```
public IActionResult Index()
{
    return View();
}
```

En la acción Index, retorno la vista principal del sitio web. Esta vista puede contener información inicial o enlaces a otras partes de la aplicación.

```
public IActionResult Privacy()
{
    return View();
}
```

La acción Privacy devuelve la vista de privacidad, la cual podría contener información sobre cómo se manejan los datos de los usuarios en el sitio web.

```
public JsonResult Login(string email, string psw_usuario)
{
    bool checkUser = false;
    ConexionBD conexionBD = new ConexionBD();
    string query = "SELECT * FROM private.usuarios WHERE email= '" + email + "' AND p
    NpgsqlCommand cmd = new NpgsqlCommand(query, conexionBD.Abrir_Conexion());

    using (NpgsqlDataReader lector = cmd.ExecuteReader())
    {
        if (lector.HasRows)
        {
            while (lector.Read())
            {
                checkUser = true;
            }
        }
    }
    string jsonResponse = "";

    if (checkUser == true)
    {
        jsonResponse = "{\"status\":200, \"data\":{\"user\":\"" + email + "\"}}";
    }
    else
    {
        jsonResponse = "{\"status\":500,\"data\":\"ERROR EL USUARIO NO EXISTE\"}";
    }
    return Json(jsonResponse);
}
```

En la acción Login, me encargo del proceso de autenticación de usuarios. Cuando un usuario intenta iniciar sesión, la información proporcionada (correo electrónico y contraseña) se verifica en la base de datos PostgreSQL. Utilizo una consulta SQL para seleccionar registros que coincidan con las credenciales proporcionadas. Si encuentro al menos un registro, asumo que el usuario es válido.
En esta acción, realizo la verificación de credenciales, creo una respuesta JSON que indica el resultado (éxito o error), y devuelvo esta respuesta al cliente que hizo la solicitud AJAX desde la página web.

```
0 referencias
public IActionResult Registrarse(string email, string psw_usuario)
{
    try
    {
        int moduloRSA = 221;
        int clavePublicaE = 5;


        List<int> numerosASCII = psw_usuario.Select(c => (int)c).ToList();
        List<int> contraseñaEncriptada = RSA.Encrypt(numerosASCII, clavePublicaE, moduloRSA);

        string contraseñaEncriptadaCadena = string.Join(",", contraseñaEncriptada);

        InsertarNuevoUsuario(email, contraseñaEncriptadaCadena);

        var jsonResponse = "{\"status\":200, \"data\":{\"user\":\"" + email + "\"}}";
        return Content(jsonResponse, "application/json");
    }
    catch (Exception ex)
    {
        var errorResponse = "{\"status\":500, \"data\":\"ERROR: " + ex.Message.Replace("\"", "
        return Content(errorResponse, "application/json");
    }
}
```

En la acción Registrarse, me encargo del proceso de registro de nuevos usuarios en el sistema. Cuando un usuario decide registrarse, se proporciona un correo electrónico y una contraseña. Antes de almacenar esta información en la base de datos PostgreSQL, aplico una capa adicional de seguridad utilizando el algoritmo RSA para encriptar la contraseña. Este proceso implica convertir la contraseña a valores ASCII, encriptarla con claves RSA predefinidas y luego almacenarla en la base de datos.En esta acción, gestiono el proceso de encriptación RSA de la contraseña antes de insertar el nuevo usuario en la base de datos. La respuesta JSON indica el resultado del registro y se envía de vuelta al cliente.

```
private void InsertarNuevoUsuario(string email, string psw_usuario)
{
    ConexionBD conexionBD = new ConexionBD();


    int moduloRSA = 221;
    int clavePublicaE = 5;

    List<int> numerosASCII = psw_usuario.Select(c => (int)c).ToList();
    List<int> contraseñaEncriptada = RSA.Encrypt(numerosASCII, clavePublicaE, moduloRSA);

    string contraseñaEncriptadaCadena = string.Join(",", contraseñaEncriptada);

    string query = "INSERT INTO private.usuarios (email, psw_usuario) VALUES (@Email, @Password)";
    NpgsqlCommand cmd = new NpgsqlCommand(query, conexionBD.Abrir_Conexion());
    cmd.Parameters.AddWithValue("@Email", email);
    cmd.Parameters.AddWithValue("@Password", contraseñaEncriptadaCadena);

    try
    {
        cmd.ExecuteNonQuery();
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error al insertar en la base de datos: {ex.Message}");
        throw;
    }
    finally
    {
        conexionBD.Cerrar_Conexion();
    }
}
```

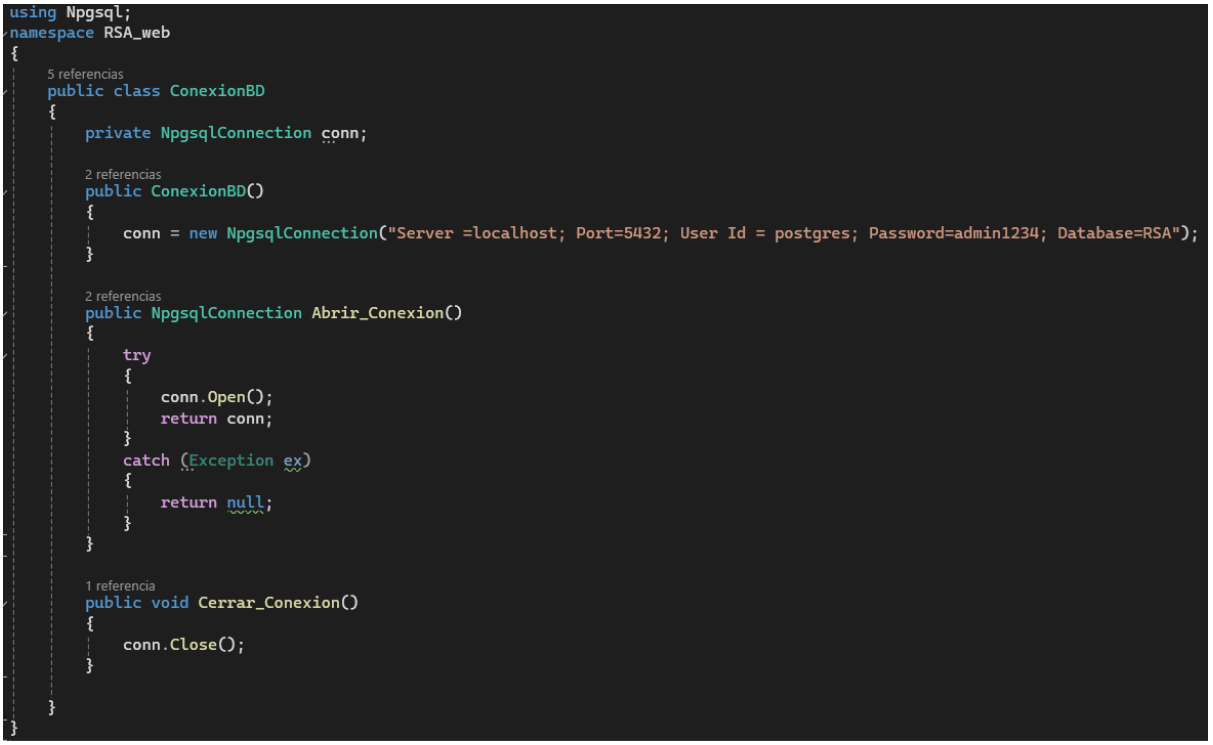
La función InsertarNuevoUsuario se encarga de insertar un nuevo usuario en la base de datos PostgreSQL. Recibe como parámetros el correo electrónico y la contraseña encriptada del usuario. Antes de realizar la inserción, se lleva a cabo una operación adicional de encriptación mediante el algoritmo RSA para garantizar la seguridad de la contraseña almacenada en la base de datos.

```
[ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore = true)]
0 referencias
public IActionResult Error()
{
    return View(new ErrorViewModel { RequestId = Activity.Current?.Id ?? HttpContext.TraceIdentifier });
}
```

El método Error maneja las solicitudes de errores y devuelve la vista correspondiente con información detallada sobre el error. Además, utiliza un atributo [ResponseCache] para indicar la configuración del almacenamiento en caché de la respuesta. En este caso, se establece que la respuesta no debe ser almacenada en caché (NoStore) y se especifica una duración de caché de 0 segundos, lo que significa que la respuesta no debe ser almacenada en caché.

Este fragmento de código garantiza que la página de error no se almacene en caché, lo que es beneficioso para mostrar información actualizada en caso de errores y problemas. El atributo [ResponseCache] proporciona control sobre cómo se almacena en caché la respuesta del servidor.

Archivo: conexionBD.cs

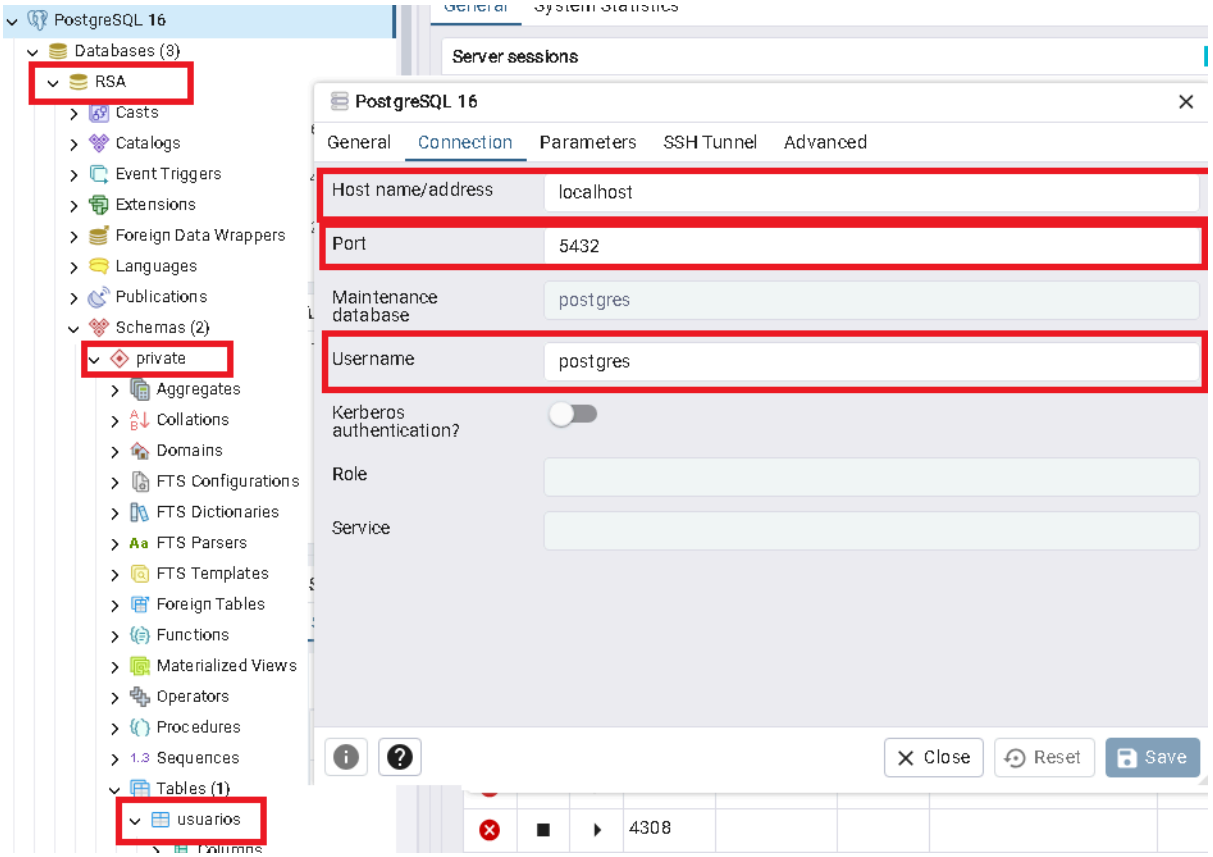


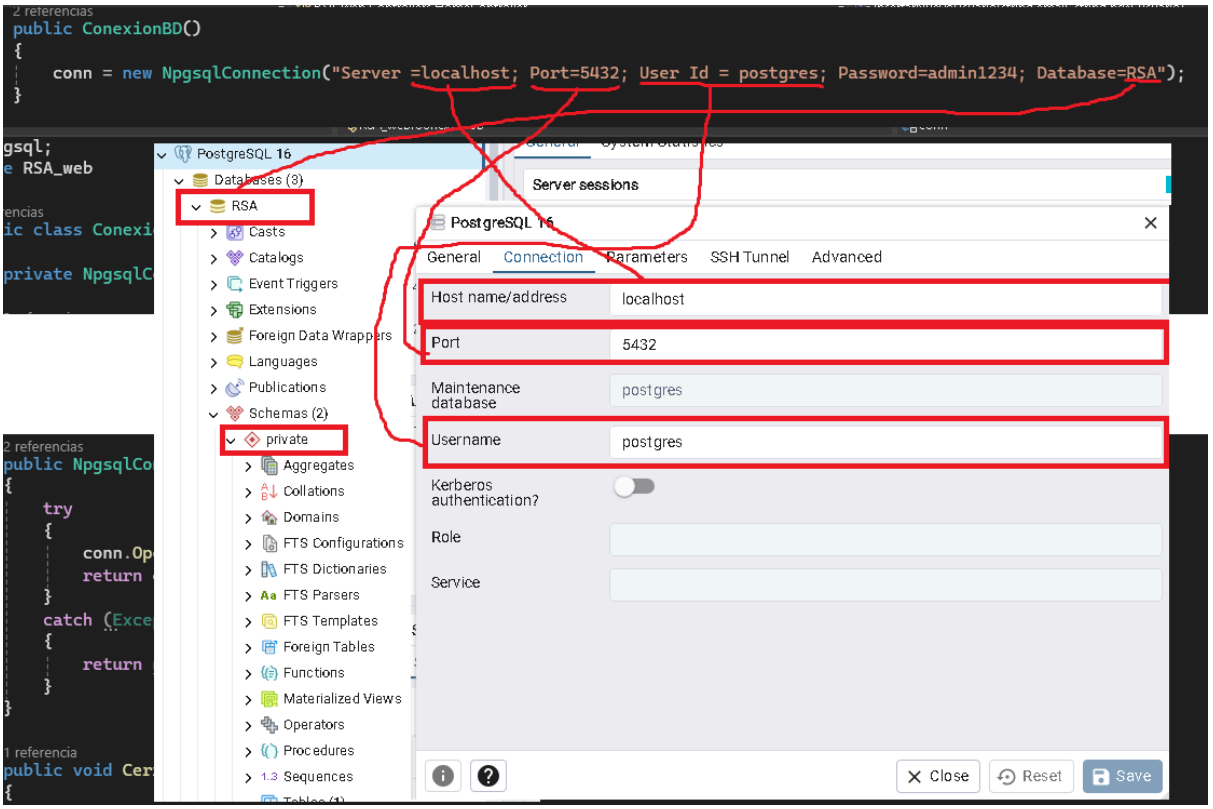
```
using Npgsql;
namespace RSA_web
{
    5 referencias
    public class ConexionBD
    {
        private NpgsqlConnection conn;

        2 referencias
        public ConexionBD()
        {
            conn = new NpgsqlConnection("Server =localhost; Port=5432; User Id = postgres; Password=admin1234; Database=RSA");
        }

        2 referencias
        public NpgsqlConnection Abrir_Conexion()
        {
            try
            {
                conn.Open();
                return conn;
            }
            catch (Exception ex)
            {
                return null;
            }
        }

        1 referencia
        public void Cerrar_Conexion()
        {
            conn.Close();
        }

    }
}
```

En el código, he definido una clase llamada ConexionBD que gestiona la conexión a la base de datos PostgreSQL utilizando Npgsql, una biblioteca específica para interactuar con bases de datos PostgreSQL en C#.

Este código encapsula la lógica para establecer y cerrar la conexión a la base de datos PostgreSQL. El constructor se encarga de configurar la cadena de conexión, y los métodos Abrir_Conexion y Cerrar_Conexion se utilizan para abrir y cerrar la conexión, respectivamente. En caso de algún problema al abrir la conexión, se devuelve null. Este diseño modular facilita la gestión de la conexión en otras partes del código.

```
2 referencias
public ConexionBD()
{
    conn = new NpgsqlConnection("Server =localhost; Port=5432; User Id = postgres; Password=admin1234; Database=RSA");
}
```




2 referencias
public ConexionBD()
{
conn = new NpgsqlConnection("Server =localhost; Port=5432; User Id = postgres; Password=admin1234; Database=RSA");
}
PostgreSQL 16
Databases (3)
RSA
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages
Publications
Schemas (2)
private
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Server sessions
PostgreSQL 16
General
Connection
Parameters
SSH Tunnel
Advanced
Host name/address
localhost
Port
5432
Maintenance database
postgres
Username
postgres
Kerberos authentication?
Role
Service
Close
Reset
Save

Archivo:RSA.cs

```
public class RSA
{
    2 referencias
    public static List<int> Encrypt(List<int> data, int publicKeyE, int modulus)
    {
        List<int> result = new List<int>();
        foreach (int value in data)
        {
            result.Add(ModPow(value, publicKeyE, modulus));
        }
        return result;
    }

    1 referencia
    private static int ModPow(int baseNum, int exponent, int modulus)
    {
        int result = 1;
        baseNum = baseNum % modulus;

        while (exponent > 0)
        {
            if (exponent % 2 == 1)
            {
                result = (result * baseNum) % modulus;
            }
            exponent = exponent >> 1;
            baseNum = (baseNum * baseNum) % modulus;
        }

        return result;
    }
}
```

En la implementación de RSA para este proyecto, he creado una clase llamada RSA que se encarga de realizar operaciones de cifrado RSA en una lista de números enteros. Esta clase tiene dos métodos principales: Encrypt y ModPow.

El método Encrypt se encarga de tomar una lista de números enteros como entrada y aplicar el cifrado RSA a cada uno de ellos. Para ello, utiliza el método ModPow, que realiza la operación de exponenciación modular, un componente clave en el algoritmo RSA.

La exponenciación modular es fundamental en RSA porque permite elevar un número a una potencia específica y calcular el residuo al dividir el resultado por un número dado. Esto es esencial para la seguridad del cifrado RSA.

Conclusión:
La verdad, esta práctica fue todo un reto. Mi primera vez trabajando con C# y bases de datos, y sí, lo intenté. Aunque la interfaz no sea la más clara visualmente, la cosa funciona. Aprendizaje puro y duro, y ahora tengo una base sólida para explorar más en este mundo del desarrollo web con ASP.NET y PostgreSQL.

A qué me refiero con que no es lo más claro visualmente, hablo de que no pude lograr que aceptara la contraseña desencriptada, es decir cuando me registraba la nueva contraseña se volvía de ley los número de encriptación

	email text	psw_usuario [PK] text
1		
2	Hermandad aRIA	121,33,121,57,51,51,57,66,121,57,175,175,57,66,33
3	honolulu	121,33,121,57,51,51,57,66,121,57,66,33,57,175,175,57,121,191,66,57,121,109,121,57,33,121,218,57,121,29,109
4	pablito	121,33,51,52,66,175,191

Si yo pongo la contraseña original 123456789 no la reconoce.

Pero si en cambio yo uso la encriptación como nueva contraseña, esto funciona.

Pero, ¿Cómo sé que esos números son RSA y no randoms?, facil, por que antes de todo trabaje en un algoritmo de consola sin interfaz ni bases de datos.

```
Cadena original: Stairway to heaven
Equivalentes num?ricos ASCII: 83 116 97 105 114 119 97 121 32 116 111 32 104 101 97 118 101 110
N?meros ASCII encriptados: 334 29 295 135 551 99 295 505 497 29 386 497 530 529 295 5 529 52
N?meros ASCII desencriptados: 83 116 97 105 114 119 97 121 32 116 111 32 104 101 97 118 101 110
Cadena recuperada: Stairway to heaven
```

```
using System;
using System.Collections.Generic;

class Program
{
    static List<int> aplicar_rsa(List<int> datos, int clave, int modulo)
    {
        List<int> resultado = new List<int>();
        foreach (int valor in datos)
        {
            resultado.Add(mod_pow(valor, clave, modulo));
        }
        return resultado;
    }

    static List<int> convertir_cadena_a_numeros_ascii(string entrada)
    {
        List<int> numeros_ascii = new List<int>();
        foreach (char c in entrada)
        {
            numeros_ascii.Add((int)c);
        }
        return numeros_ascii;
    }

    static string convertir_numeros_ascii_a_cadena(List<int> numeros_ascii)
    {
        return new string(numeros_ascii.ConvertAll(c => (char)c).ToArray());
    }

    static int mod_pow(int base_num, int exponente, int modulo)
    {
        int resultado = 1;
        base_num = base_num % modulo;

        while (exponente > 0)
        {
            if (exponente % 2 == 1)
            {
                resultado = (resultado * base_num) % modulo;
            }
            exponente = exponente >> 1;
            base_num = (base_num * base_num) % modulo;
        }

        return resultado;
    }

    static void Main()
```

```
    {
        // Claves RSA para el ejemplo
        int modulo = 589;
        int clave_publica_e = 17;
        int clave_privada_d = 413;

        // Ejemplo de uso:
        string ejemplo_cadena = "Stairway to heaven";
        List<int> numeros_ascii = convertir_cadena_a_numeros_ascii(ejemplo_cadena);

        // Mostrar resultados
        Console.WriteLine("Cadena original: " + ejemplo_cadena);
        Console.Write("Equivalentes numéricos ASCII: ");
        foreach (int num in numeros_ascii)
        {
            Console.Write(num + " ");
        }
        Console.WriteLine();

        // Encriptar los números ASCII
        List<int> numeros_encriptados = aplicar_rsa(numeros_ascii, clave_publica_e, modulo);
        Console.Write("Números ASCII encriptados: ");
        foreach (int num in numeros_encriptados)
        {
            Console.Write(num + " ");
        }
        Console.WriteLine();

        // Desencriptar los números ASCII
        List<int> numeros_desencriptados = aplicar_rsa(numeros_encriptados, clave_privada_d, modulo);
        Console.Write("Números ASCII desencriptados: ");
        foreach (int num in numeros_desencriptados)
        {
            Console.Write(num + " ");
        }
        Console.WriteLine();

        // Convertir los números ASCII desencriptados de vuelta a cadena
        string cadena_recuperada = convertir_numeros_ascii_a_cadena(numeros_desencriptados);
        Console.WriteLine("Cadena recuperada: " + cadena_recuperada);
    }

    static int calcular_inverso(int a, int m)
    {
        // Algoritmo extendido de Euclides para calcular el inverso modular
        int m0 = m, t, q;
        int x0 = 0, x1 = 1;

        while (a > 1)
        {
            q = a / m;
            t = m;

            m = a % m;
            a = t;
            t = x0;

            x0 = x1 - q * x0;
            x1 = t;
        }

        if (x1 < 0)
            x1 += m0;

        return x1;
    }

    static (int clave, int modulo) generar_clave_rsa(int p, int q, int e)
    {
        var clave_publica = (p * q, e);
        return clave_publica;
    }

    static void imprimir_numeros_ascii(List<int> numeros_ascii)
    {
        Console.Write("Números ASCII: ");
        foreach (int num in numeros_ascii)
        {
            Console.Write(num + " ");
        }
        Console.WriteLine();
    }
}
```

Referencias:

Coding con C. (2021a, September 21). *Cómo INSTALAR una BASE de DATOS (2022) ➤ PostgreSQL ➤ Instalar PostGIS* [Video]. YouTube. https://www.youtube.com/watch?v=PkrCnoOdqdw

Coding con C. (2021b, November 15). *Conexión a BD y sentencias SQL en C# ➤ CURSO en C#* [Video]. YouTube. https://www.youtube.com/watch?v=gAodyg1ZlyI

Coding con C. (2021c, November 15). *Conexión a BD y sentencias SQL en C# ➤ CURSO en C#* [Video]. YouTube. https://www.youtube.com/watch?v=gAodyg1ZlyI

Coding con C. (2023a, January 13). *Cómo trabajar las respuestas JSON de una API en C#* [Video]. YouTube. https://www.youtube.com/watch?v=70WoDE9UxFI

Coding con C. (2023b, February 10). *Desarrollo de un sistema de inicio de sesión en MVC con C# paso a paso* [Video]. YouTube. https://www.youtube.com/watch?v=qtfG8-eye5U

Juan Manuel Ramirez. (2018a, January 12). *CRIPTOGRAFIA:METODO RSA (EXPLICADO CON MIS PROPIAS PALABRAS) PARTE 1.* [Video]. YouTube. https://www.youtube.com/watch?v=AjaMZddJlK0

Juan Manuel Ramirez. (2018b, January 12). *CRIPTOGRAFIA:METODO RSA (EXPLICADO CON MIS PROPIAS PALABRAS) PARTE 2.* [Video]. YouTube. https://www.youtube.com/watch?v=AliXPLkzxJE

Juan Manuel Ramirez. (2018c, January 13). *CRIPTOGRAFIA:METODO RSA (EXPLICADO CON MIS PROPIAS PALABRAS) PARTE 3.* [Video]. YouTube. https://www.youtube.com/watch?v=Pl7dfr-GYDE