# Reporte de Vulnerabilidad

**Archivo:** UserController.cs

**Código Analizado:**

```
?using AutoMapper;
using FermaOrders.API.Controllers.Response;
using FermaOrders.Application.DTOs.Auth;
using FermaOrders.Application.Interface.Auth;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System.Net;
using System.Web.Http;
using FromBodyAttribute = Microsoft.AspNetCore.Mvc.FromBodyAttribute;
using HttpPostAttribute = Microsoft.AspNetCore.Mvc.HttpPostAttribute;
using RouteAttribute = Microsoft.AspNetCore.Mvc.RouteAttribute;

namespace FermaOrders.API.Controllers.Auth
{
    [Route("api/[controller]")]
    [ApiController]
    public class UserController : ControllerBase
    {
        private readonly IWebHostEnvironment _env;
        private readonly IUsuarioService _usuarioService;
        private readonly IMapper _mapper;
        protected RespuestaAPI _respuestaAPI;

        public UserController(IUsuarioService usuarioService, IMapper mapper,
IWebHostEnvironment env)
        {
            _usuarioService = usuarioService;
            _mapper = mapper;
            _respuestaAPI = new();
            _env = env;
        }

        [AllowAnonymous]
        [HttpPost("Registro")]
        [ProducesResponseType(StatusCodes.Status201Created)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        public async Task<IActionResult> Registro([FromForm] UsuarioRegistroDto
usuarioRegistroDto)
        {

            if (!ModelState.IsValid)
            {
                _respuestaAPI.StatusCode = HttpStatusCode.BadRequest;
                _respuestaAPI.IsSuccess = false;
                _respuestaAPI.ErrorMessages.Add("Datos de registro inválidos");
                return BadRequest(_respuestaAPI);
            }


            if (!_usuarioService.IsUniqueUser(usuarioRegistroDto.NombreUsuario))
            {
                _respuestaAPI.StatusCode = HttpStatusCode.BadRequest;
                _respuestaAPI.IsSuccess = false;
                _respuestaAPI.ErrorMessages.Add("El nombre de usuario ya existe");
                return BadRequest(_respuestaAPI);
            }

            var usuarioCreado = await _usuarioService.RegistroAsync(usuarioRegistroDto);
            if (usuarioCreado == null)
            {
                _respuestaAPI.StatusCode = HttpStatusCode.InternalServerError;
                _respuestaAPI.IsSuccess = false;
                _respuestaAPI.ErrorMessages.Add("Error en el registro de usuario");
                return StatusCode(StatusCodes.Status500InternalServerError, _respuestaAPI);
            }


            _respuestaAPI.StatusCode = HttpStatusCode.Created;
```

```
            _respuestaAPI.IsSuccess = true;
            _respuestaAPI.Result = usuarioCreado;
            return CreatedAtAction(nameof(Registro), new { id = usuarioCreado.Id },
_respuestaAPI);
        }


        [AllowAnonymous]
        [HttpPost("Login")]
        [ProducesResponseType(StatusCodes.Status201Created)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        public async Task<IActionResult> Login([FromForm] UsuarioLoginDto usuarioLoginDto)
        {
            var respuestaLogin = await _usuarioService.LoginAsync(usuarioLoginDto);

            if (respuestaLogin == null || string.IsNullOrEmpty(respuestaLogin.Token))
            {
                _respuestaAPI.StatusCode = HttpStatusCode.BadRequest;
                _respuestaAPI.IsSuccess = false;
                _respuestaAPI.ErrorMessages.Add("El nombre de Usuario o Password son
Incorrectos");
                return BadRequest(_respuestaAPI);
            }

            _respuestaAPI.StatusCode = HttpStatusCode.OK;
            _respuestaAPI.IsSuccess = true;
            ///Usamos esto para que nos devuelva el login
            _respuestaAPI.Result = respuestaLogin;
            return Ok(_respuestaAPI);
        }

        [Authorize]
        [HttpPost("logout")]
        public async Task<IActionResult> Logout()
        {
            var token = Request.Headers["Authorization"].ToString().Replace("Bearer ", "");

            await _usuarioService.LogoutAsync(token);

            return Ok(new { message = "Sesión cerrada correctamente." });
        }
    }
}
```

**Análisis:** ```html
Análisis de Seguridad y Calidad del Código
Vulnerabilidades de Seguridad
Vulnerabilidad 1: Falta de validación y sanitización robusta en `UsuarioRegistroDto`

##### Descripción

Aunque se usa `ModelState.IsValid`, esto depende de los atributos de validación en el DTO. Una validación insuficiente en el DTO o la ausencia de sanitización adecuada puede llevar a inyección de código o Cross-Site Scripting (XSS) si los datos del usuario se muestran en alguna parte de la aplicación.

##### Línea Aproximada

Líneas relacionadas con el uso de `usuarioRegistroDto` en el método `Registro`, especialmente al pasar datos al servicio (aprox. línea 43).

##### Mitigación

- Implementar validaciones robustas en el `UsuarioRegistroDto`, incluyendo la longitud máxima de las cadenas, formatos esperados (ej: email) y listas blancas (ej: para caracteres permitidos).
- Sanitizar los datos antes de almacenarlos en la base de datos, especialmente si se van a mostrar en otras partes de la aplicación. Considerar el uso de bibliotecas de sanitización.
- Utilizar un filtro global para validar todos los modelos entrantes y evitar la repetición de código.
```
```

Vulnerabilidad 2: Almacenamiento inseguro de contraseñas (implícito)

#### Descripción

El código proporcionado no muestra cómo se manejan las contraseñas, pero es crucial asegurarse de que no se almacenan en texto plano. Si se almacenan contraseñas sin un hash seguro y un salt, la aplicación es vulnerable a robo de credenciales en caso de una brecha de seguridad.

#### Línea Aproximada

Implícito en el servicio `_usuarioService`, específicamente en el método `RegistroAsync` (aprox. línea 51) y `LoginAsync` (aprox. línea 72).

#### Mitigación

- Utilizar una función de hashing segura (como bcrypt, scrypt o Argon2) con un salt único para cada contraseña.
- Asegurarse de que la biblioteca de hashing está actualizada para evitar vulnerabilidades conocidas.

Vulnerabilidad 3: Vulnerabilidad a ataques CSRF en el endpoint de Login

#### Descripción

El endpoint de Login utiliza [FromForm], lo que lo hace susceptible a ataques CSRF. Un atacante podría forzar al usuario a enviar una solicitud maliciosa desde otro sitio web.

#### Línea Aproximada

Línea 70 `public async Task Login([FromForm] UsuarioLoginDto usuarioLoginDto)`

#### Mitigación

- Usar el atributo [FromBody] en lugar de [FromForm] para los endpoints de Login.
- Implementar tokens CSRF en el frontend y validarlos en el backend.

Vulnerabilidad 4: Falta de validación del token de autorización al cerrar sesión

#### Descripción

En el método Logout, se extrae el token del encabezado "Authorization" sin validarlo. Un atacante podría falsificar o manipular el token, lo que podría llevar a problemas de seguridad. Aunque se llama a `_usuarioService.LogoutAsync(token)`, la validación real del token debe ocurrir *antes* de llamar a este servicio para prevenir el procesamiento de tokens inválidos.

#### Línea Aproximada

Línea 91: `var token = Request.Headers["Authorization"].ToString().Replace("Bearer ", "");`

#### Mitigación

- Validar el formato del token (por ejemplo, si es un JWT, validar su estructura y firma).
- Comprobar si el token existe en una lista de tokens válidos/activos.
- Verificar si el usuario asociado al token tiene permisos para realizar la operación de cierre de sesión.

Métricas de Calidad del Código
Complejidad Ciclomática

#### Análisis

La complejidad ciclomática general parece baja en este fragmento, ya que los métodos son relativamente cortos y con pocas ramas condicionales. Sin embargo, la lógica dentro de `_usuarioService` podría ser más compleja y requerir un análisis separado.

#### Mejora

- Refactorizar métodos largos en funciones más pequeñas y reutilizables si la lógica interna de `_usuarioService` es compleja.
- Usar principios SOLID (especialmente SRP - Single Responsibility Principle) para asegurar que cada clase y método tiene una única responsabilidad.

Duplicación

#### Análisis

Hay cierta duplicación en la construcción de la respuesta `_respuestaAPI`. Se repite la asignación de `StatusCode`, `IsSuccess` y la adición de mensajes de error en múltiples lugares.

#### Mejora

- Crear un método auxiliar para construir el objeto `_respuestaAPI` con base en el estado y mensaje de error. Esto centraliza la lógica y reduce la duplicación. Por ejemplo, métodos como `SetBadRequestResponse(string message)` o `SetInternalServerErrorResponse(string message)`.
- Considerar el uso de un filtro de excepciones global para manejar errores y construir la respuesta de error centralizadamente.

Legibilidad

#### Análisis

El código es generalmente legible debido a nombres de variables descriptivos y una estructura clara. El uso de comentarios es mínimo, pero el código es lo suficientemente auto-explicativo en la mayoría de los casos.

#### Mejora

- Añadir comentarios en secciones de código que puedan ser menos obvias o que realicen operaciones complejas.
- Mantener la consistencia en el estilo de codificación (espacios, nombres de variables, etc.).

Acoplamiento

#### Análisis

El controlador tiene un acoplamiento moderado con `IUsuarioService`, `IMapper` y `IWebHostEnvironment`. Esto es esperable en un controlador, pero es importante asegurarse de que la interfaz `IUsuarioService` esté bien definida y siga el principio de Inversión de Dependencia (DIP).

#### Mejora

- Asegurarse de que `IUsuarioService` define una abstracción clara y concisa de las operaciones relacionadas con el usuario.
- Evitar la inyección directa de dependencias que no son absolutamente necesarias en el controlador. Considerar el uso de un patrón Mediator o similar si el controlador necesita interactuar con múltiples servicios.

Solución Propuesta (Fragmento de Ejemplo)

Este es un ejemplo de cómo se podría abordar algunas de las vulnerabilidades y mejorar la calidad del código. Este es un fragmento y requeriría adaptación al resto de la aplicación.

```
using AutoMapper;
using FermaOrders.Application.DTOs.Auth;
using FermaOrders.Application.Interface.Auth;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using System.Net;
using System.Security.Cryptography;
using System.Text;
using System.Threading.Tasks;

namespace FermaOrders.API.Controllers.Auth
{
    [Route("api/[controller]")]
    [ApiController]
    public class UserController : ControllerBase
    {
        private readonly IUsuarioService _usuarioService;
        private readonly IMapper _mapper;
        private readonly ILogger<UserController> _logger;  // Add logger
        protected RespuestaAPI _respuestaAPI;

        public UserController(IUsuarioService usuarioService, IMapper mapper,
ILogger<UserController> logger)
        {
            _usuarioService = usuarioService;
            _mapper = mapper;
            _logger = logger;
            _respuestaAPI = new();
        }

        [AllowAnonymous]
        [HttpPost("Registro")]
        [ProducesResponseType(StatusCodes.Status201Created)]
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        public async Task<IActionResult> Registro([FromBody] UsuarioRegistroDto
usuarioRegistroDto) // Use [FromBody]
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(CreateErrorResponse("Datos de registro inválidos"));
            }

            if (!_usuarioService.IsUniqueUser(usuarioRegistroDto.NombreUsuario))
            {
                return BadRequest(CreateErrorResponse("El nombre de usuario ya existe"));
            }

            try
            {
                var usuarioCreado = await _usuarioService.RegistroAsync(usuarioRegistroDto);
                if (usuarioCreado == null)
                {
                    _logger.LogError("Error al registrar el usuario {Username}",
usuarioRegistroDto.NombreUsuario);
                    return StatusCode(StatusCodes.Status500InternalServerError,
CreateErrorResponse("Error en el registro de usuario"));
                }

                _respuestaAPI.StatusCode = HttpStatusCode.Created;
                _respuestaAPI.IsSuccess = true;
                _respuestaAPI.Result = usuarioCreado;
                return CreatedAtAction(nameof(Registro), new { id = usuarioCreado.Id },
_respuestaAPI);
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Excepción no controlada al registrar el usuario
```

```
{Username}", usuarioRegistroDto.NombreUsuario);
                return StatusCode(StatusCodes.Status500InternalServerError,
CreateErrorResponse("Error inesperado en el registro"));
            }
        }

        [AllowAnonymous]
        [HttpPost("Login")]
        [ProducesResponseType(StatusCodes.Status200OK)] // Use 200 OK for successful login
        [ProducesResponseType(StatusCodes.Status400BadRequest)]
        [ProducesResponseType(StatusCodes.Status500InternalServerError)]
        public async Task<IActionResult> Login([FromBody] UsuarioLoginDto usuarioLoginDto)  //
Use [FromBody]
        {
            if (!ModelState.IsValid)
            {
                 return BadRequest(CreateErrorResponse("Datos de inicio de sesión
inválidos"));
            }

            try
            {
                var respuestaLogin = await _usuarioService.LoginAsync(usuarioLoginDto);

                if (respuestaLogin == null || string.IsNullOrEmpty(respuestaLogin.Token))
                {
                    return BadRequest(CreateErrorResponse("El nombre de Usuario o la
Contraseña son Incorrectos"));
                }

                _respuestaAPI.StatusCode = HttpStatusCode.OK;
                _respuestaAPI.IsSuccess = true;
                _respuestaAPI.Result = respuestaLogin;
                return Ok(_respuestaAPI); // Use Ok instead of Created
            }
            catch (Exception ex)
            {
                _logger.LogError(ex, "Error al iniciar sesión el usuario {Username}",
usuarioLoginDto.NombreUsuario);
                return StatusCode(StatusCodes.Status500InternalServerError,
CreateErrorResponse("Error inesperado al iniciar sesión"));
            }
        }

        [Authorize]
        [HttpPost("logout")]
        public async Task<IActionResult> Logout()
        {
            var token = Request.Headers["Authorization"].ToString().Replace("Bearer ", "");

            // Validate Token format BEFORE passing to the service
            if (string.IsNullOrEmpty(token) || !IsValidJwtToken(token))
            {
                return BadRequest(CreateErrorResponse("Token de autorización inválido."));
            }


            await _usuarioService.LogoutAsync(token);

            return Ok(new { message = "Sesión cerrada correctamente." });
        }


        private IActionResult CreateErrorResponse(string message)
        {
            _respuestaAPI.StatusCode = HttpStatusCode.BadRequest;
            _respuestaAPI.IsSuccess = false;
            _respuestaAPI.ErrorMessages = new List<string> { message }; // Ensure
ErrorMessages is initialized
            return BadRequest(_respuestaAPI);
        }

         private bool IsValidJwtToken(string token)
        {
            // Implement a proper JWT validation logic here.
            // This is a placeholder. You should validate the token's signature,
            // expiration, issuer, audience, etc.  Consider using a library
            // like System.IdentityModel.Tokens.Jwt for full JWT validation.
```

```
            // Example (basic structure check, replace with proper validation):
            return !string.IsNullOrWhiteSpace(token) && token.Split('.').Length == 3;
        }
    }
}
```

- **Validación de Modelo y Sanitización:** Se usa `[FromBody]` en lugar de `[FromForm]` en los métodos `Registro` y `Login` para mitigar ataques CSRF. Se ha añadido validación básica de formato del token en el método `Logout`.
- **Manejo de Excepciones:** Se ha añadido un bloque `try-catch` para capturar excepciones y registrarlas con un logger.
- **Duplicación de Código:** Se ha creado un método auxiliar `CreateErrorResponse` para evitar la duplicación en la creación de la respuesta de error.
- **Logging:** Se ha añadido un logger para registrar errores y excepciones.
- **Validacion del token:** Implementado un metodo para validar el token antes de cerrar la sesión

```
```