Documentación Profesional de la API REST

PreventaBBO

Módulo de Autenticación y Campañas

Generado por *ROIDER* (MILLARES)
Fecha: 31 de octubre de 2025

Índice

1 Introducción

La API REST de **PreventaBBO** está desarrollada en `.NET 8` con arquitectura limpia, utilizando `ASP.NET Core`, `Entity Framework Core`, `JWT` para autenticación y `SignalR` para notificaciones en tiempo real.

Este documento se centra en:

- **Lógica de autenticación CFE** (login, cambio de contraseña, gestión de sesiones).
- **Módulo de campañas** con control de acceso por rol `CAMPAÑA`.
- Endpoints REST, DTOs, servicios y repositorios.

La autenticación es segura, auditable y configurable mediante parámetros en base de datos.

2 Arquitectura de Autenticación

Controlador `CfeAuthController` → Maneja solicitudes HTTP.

Servicio `CfeAuthenticationService` → Lógica de negocio.

Repositorio `SgtUsuarioRepository` → Acceso a datos.

Base de datos Tablas: `SgtUsuarios`, `SgtSesiones`, `SgtAuditoria*`.

Se utiliza **JWT** firmado con `HMAC-SHA256`, con roles en claims.

3 Endpoint: Login (`POST /api/CfeAuth/login`)

3.1 Descripción

Autentica al usuario, genera JWT y registra sesión. Soporta cambio obligatorio de contraseña.

3.2 Request

```
{
    "username": "jdoe",
    "password": "P@ssw0rd2025!"
}
```

Listing 1: Ejemplo de solicitud

3.3 Response Exitosa (200 OK)

```
{
    "success": true,
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6...",
    "expiration": "2025-10-27T14:30:00Z",
    "userInfo": {
        "username": "jdoe",
        "displayName": "John Doe",
        "email": "jdoe@empresa.com",
        "roles": ["ADMIN", "CAMPA A"]
    }
}
```

Listing 2: Respuesta con token

3.4 Código del Controlador

```
[HttpPost("login")]
[AllowAnonymous]
[ProducesResponseType(typeof(AuthenticationResultDto), 200)]
[ProducesResponseType(typeof(AuthenticationResultDto), 401)]
public async Task<IActionResult> Login([FromBody] AuthenticationRequestDto
    request)
{
    try
    {
        var result = await _authService.AuthenticateAsync(request.Username,
    request.Password);
        if (!result.Success) return Unauthorized(result);

        if (result.RequiresPasswordChange)
        {
            return Ok(new
            {
                result.Success,
                result.UserInfo,
                RequiresPasswordChange = true,
                Message = "Debe cambiar su contrase a antes de continuar"
            });
        }

        return Ok(result);
    }
    catch (Exception ex)
    {
        return StatusCode(500, new { message = $"Error en autenticaci n: {
    ex.Message}" });
    }
}
```

Listing 3: CfeAuthController.Login

4 Lógica del Servicio: `AuthenticateAsync`

1. Obtiene IP y dispositivo (User-Agent parsing).
2. Valida credenciales en repositorio.
3. Verifica si `UsrCambiaContra = true` → requiere cambio.
4. Genera JWT con roles.
5. Cierra sesiones previas si `PERMITIR_SESIONES_CONCURRENTES = false`.
6. Registra nueva sesión en `SgtSesiones`.
7. Envía notificación via SignalR.

```
public async Task<AuthenticationResultDto> AuthenticateAsync(string username
    , string password)
{
    var result = new AuthenticationResultDto();
    var ipAddress = GetIpAddress();

```

```
    var (isValid, error) = await _usuarioRepository.ValidateCredentialsAsync
    (username, password, ipAddress);
    if (!isValid) { result.Success = false; result.Errors.Add(error); return
     result; }

    var userInfo = await GetUserInfoAsync(username);
    var usuario = await _usuarioRepository.GetByUsernameAsync(username);

    if (usuario?.UsrCambiaContra == true)
    {
        result.RequiresPasswordChange = true;
        result.UserInfo = userInfo;
        return result;
    }

    var token = GenerateJwtToken(userInfo);
    await RegistrarSesionAsync(username, token);

    result.Success = true;
    result.Token = token;
    result.Expiration = DateTime.UtcNow.AddMinutes(_jwtExpirationMinutes);
    result.UserInfo = userInfo;

    return result;
}
```

Listing 4: AuthenticateAsync (resumen)

5 Validación de Credenciales en Repositorio

```
public async Task<(bool IsValid, string? ErrorMessage)>
    ValidateCredentialsAsync(string username, string password, string
    ipAddress)
{
    var usuario = await _context.SgtUsuarios.FirstOrDefaultAsync(u => u.
    UsrId == username);
    if (usuario == null) return (false, "Usuario no encontrado");

    if (usuario.UsrEstado != "A") return (false, "Usuario inactivo");

    if (usuario.UsrFechaBloqueoPorIntentos > DateTime.Now)
        return (false, "Usuario bloqueado por intentos fallidos");

    bool isValid = VerifyPassword(password, usuario.UsrContrasena, usuario.
    UsrSalt);
    if (!isValid)
    {
        usuario.UsrIntentosFallidos++;
        if (usuario.UsrIntentosFallidos >= await
    ObtenerParametroSeguridadIntAsync("MAX_INTENTOS_LOGIN", 3))
        {
            usuario.UsrFechaBloqueoPorIntentos = DateTime.Now.AddMinutes(
    await ObtenerParametroSeguridadIntAsync("TIEMPO_BLOQUEO_MINUTOS", 30));
        }
        await _context.SaveChangesAsync();
        await RegistrarIntentoLoginAsync(username, false, "Contrase a
    incorrecta", ipAddress);
        return (false, "Contrase a incorrecta");
    }

    usuario.UsrIntentosFallidos = 0;
```

```
25      await _context.SaveChangesAsync();
26      await RegistrarIntentoLoginAsync(username, true, null, ipAddress);
27      return (true, null);
28  }
```

Listing 5: ValidateCredentialsAsync

6 Otros Endpoints de Autenticación

Endpoint	Método	Descripción
`/change-password`	POST	Cambia contraseña (requiere auth)
`/validate-token`	POST	Valida JWT y sesión activa
`/user-info`	GET	Información del usuario autenticado
`/register`	POST	Registro de nuevo usuario
`/request-password-reset`	POST	Solicita token de recuperación
`/reset-password`	POST	Restablece contraseña con token
`/logout`	POST	Cierra sesión (SignalR)
`/change-password-initial`	POST	Cambio inicial sin auth

7 Módulo de Campañas

El acceso al módulo de campañas está restringido por el rol `CAMPAÑA`.

7.1 Verificación de Rol

```
1  private const string ROL_CAMPANA = "CAMPA A";
2
3  public async Task<bool> HasCampanaRoleAsync(string username)
4  {
5      return await HasRoleAsync(username, ROL_CAMPANA);
6  }
```

Listing 6: HasCampanaRoleAsync

7.2 Middleware (opcional)

Se puede usar un middleware personalizado:

```
1  app.UseCampanaRoleMiddleware(); // Comentado en startup
```

8 Seguridad y Configuración

8.1 Reglas de Contraseña (configurables)

Parámetro	Valor por defecto
LONGITUD_MIN_CONTRASENA	12
REQUIERE_MAYUSCULAS	true
REQUIERE_NUMEROS	true
REQUIERE_CARACTERES_ESPECIALES	true
HISTORIAL_CONTRASENAS	6
PERMITIR_SESIONES_CONCURRENTES	false

Cuadro 2: Parámetros de seguridad en `SgtParametrosSeguridads`

8.2 JWT Configuration

```
"JwtSettings": {
    "Secret": "tu_clave_secreta_muy_larga_y_segura_1234567890",
    "Issuer": "PreventaBBO",
    "Audience": "PreventaBBO-Client",
    "ExpirationMinutes": 60,
    "InactivityTimeoutMinutes": 15
}
```

Listing 7: appsettings.json

9 Conclusión

La API de **PreventaBBO** ofrece un sistema de autenticación robusto, seguro y auditable, con soporte para:

- Autenticación CFE con control de sesiones.
- Gestión de roles (incluyendo `CAMPAÑA`).
- Auditoría completa.
- Integración con SignalR.
- Configuración flexible.

Ideal para entornos empresariales con altos estándares de seguridad.