SafeGoalStats
Daniel Pariente | Alex Rubio

Medidas de seguridad

1. Helmet

Vamos a añadir una herramienta llamada **Helmet**.
Su función es sencilla: **poner una capa extra de seguridad automáticamente**.

Esto ayuda a proteger la web frente a cosas como:

- que otra página intente cargar tu web dentro de un marco falso para engañar al usuario;
- que el navegador interprete archivos de forma incorrecta;
- ciertos ataques donde se intenta inyectar código malicioso en páginas web.

No hace milagros, pero **es una medida básica muy recomendable** y bastante usada.

```
/var/www/safegoalstats-api$ npm install helmet
```

2. Cerrar Cors

Ahora vamos a hacer que nuestro backend deje de aceptar peticiones de cualquier web, esto quiere decir que cualquier página puede usar nuestra api, y lo que vamos a hacer es que sólo nuestra web pueda conectarse.

Tambien vamos a poner un limite de intentos, que esto va a evitar que se sature la **api**, que se intenten logins masivos o que se pueda probar contraseñas sin parar

```
usuario@ubuntu-2404-desktop:/var/www/safegoalstats-api$ npm install express-rate-lim
it
```

.

SafeGoalStats
Daniel Pariente | Alex Rubio

```
const express = require('express');
const cors = require('cors');
const helmet = require('helmet');
const rateLimit = require('express-rate-limit');
require('dotenv').config();

const db = require('./database');
const authRoutes = require('./routes/authRoutes');
const footballRoutes = require('./routes/footballRoutes');

const app = express();


// Seguridad básica del servidor
app.use(helmet());


// Limitar número de peticiones
const limiter = rateLimit({
    windowMs: 15 * 60 * 1000,
    max: 100,
    message: {
        error: 'Has realizado demasiadas peticiones. Inténtalo de nuevo dentro de unos minutos.'
    }
});

app.use(limiter);


// Permitir únicamente conexiones desde SafeGoalStats
app.use(cors({
    origin: 'http://192.168.239.119'
}));


app.use(express.json());


// Rutas
app.use('/api/auth', authRoutes);
app.use('/api', footballRoutes);
```

```
42
43
44  // Ruta base
45  app.get('/', (req, res) => {
46      res.json({
47          mensaje: 'SafeGoalStats API funcionando correctamente'
48      });
49  });
50
51
52  const PORT = process.env.PORT || 3000;
53
54  app.listen(PORT, () => {
55      console.log(`Servidor corriendo en http://localhost:${PORT}`);
56  });
```

3. Validación de formularios

Se añadió un sistema de comprobación de datos en registro y login para asegurar que la información introducida cumple unos requisitos mínimos. De esta forma se evitan correos inválidos, contraseñas demasiado débiles y nombres de usuario incorrectos, mejorando tanto la seguridad como la calidad de los datos almacenados.

```
usuario@ubuntu-2404-desktop:/var/www/safegoalstats-apt$ npm install express-validato
r

added 3 packages, and audited 128 packages in 1s

33 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
usuario@ubuntu-2404-desktop:/var/www/safegoalstats-apt$
```

```
const { body, validationResult } = require('express-validator');

const validarRegistro = [
    body('username')
        .trim()
        .isLength({ min: 3 })
        .withMessage('El usuario debe tener mínimo 3 caracteres'),

    body('email')
        .isEmail()
        .normalizeEmail()
        .withMessage('Introduce un email válido'),

    body('password')
        .isLength({ min: 8 })
        .withMessage('La contraseña debe tener mínimo 8 caracteres'),

    (req, res, next) => {
        const errores = validationResult(req);

        if (!errores.isEmpty()) {
            return res.status(400).json({
                errores: errores.array()
            });
        }

        next();
    }
];
```

```
const validarLogin = [
    body('email')
        .isEmail()
        .normalizeEmail()
        .withMessage('Email inválido'),

    body('password')
        .notEmpty()
        .withMessage('Debes introducir contraseña'),

    (req, res, next) => {
        const errores = validationResult(req);

        if (!errores.isEmpty()) {
            return res.status(400).json({
                errores: errores.array()
            });
        }

        next();
    }
];

module.exports = {
    validarRegistro,
    validarLogin
};
```

SafeGoalStats
Daniel Pariente | Alex Rubio

Qué hace esto

.trim()

Quita espacios: antes: daniel después:daniel

.isLength({min:3})

mínimo 3 letras.

Evita usuarios absurdos.

.isEmail()

comprueba formato correcto:

correo@gmail.com
 hola123

.normalizeEmail()

limpia email:

DANIEL@GMAIL.COM

↓

daniel@gmail.com

mejor para comparar.

contraseña mínima 8 caracteres

validationResult

Recoge errores.

Si algo está mal:

→ bloquea petición.

No entra al controlador.

4. Activar las validaciones en authRoutes.js

Aunque hemos creado **validarRegistro y validarLogin** estos nunca se ejecutan, paara ello tenemos que meterlos en el archivo de **authRoutes.js**

Qué hace esto

Antes:

petición → controlador

Ahora:

petición → validación → controlador

Si algo está mal:

petición → validación → BLOQUEADA

No llega ni al login ni al registro.

5. Crear middleare JWT real

Otro problema es que ahora mismo tenemos tokens pero no protegen rutas backend, básicamente quiere decir que cualquiera puede pedir los datos de **estadísticas,partidos y clasificación** sin necesidad de iniciar sesión.

SafeGoalStats
Daniel Pariente | Alex Rubio

src > middleware > JS verificarToken.js > ...

```
const jwt = require('jsonwebtoken');

function verificarToken(req, res, next) {

    const authHeader = req.headers.authorization;

    if (!authHeader) {
        return res.status(401).json({
            error: 'Acceso denegado. No hay token.'
        });
    }

    const token = authHeader.split(' ')[1];

    try {
        const verificado = jwt.verify(
            token,
            process.env.JWT_SECRET
        );

        req.usuario = verificado;
        next();

    } catch (error) {
        return res.status(403).json({
            error: 'Token inválido o caducado'
        });
    }
}

module.exports = verificarToken;
```

Pondremos esto que se encarga de

Busca cabecera:

Authorization: Bearer TOKEN

Si existe:

→ comprueba firma.

Si el token es bueno:

→ deja pasar.

Si no:

→ bloquea.

En sencillo

Antes:

cualquiera entra.

Ahora:

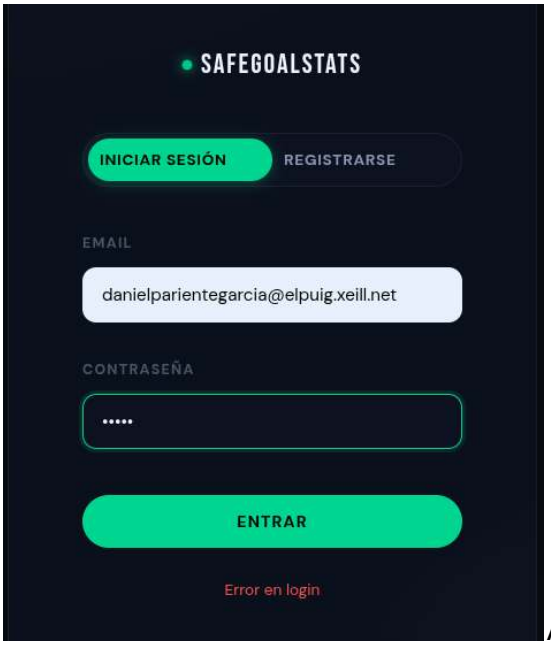
solo usuarios logueados.

6. Añadir logs basicos en login

Ahora vamos a añadir algunos **console.log**

```
// Buscar el usuario
const usuario = await Usuario.buscarPorEmail(email);
if (!usuario) {
    console.log("Login fallido | usuario no encontrado:", email);
    console.log("Login fallido | contraseña incorrecta:", email);
    console.log("Login correcto:", email);
```

Que estos se van a encargar de si el usuario no se encuentra, de cuando falla la contraseña, y de cuando cuando el login es correcto.



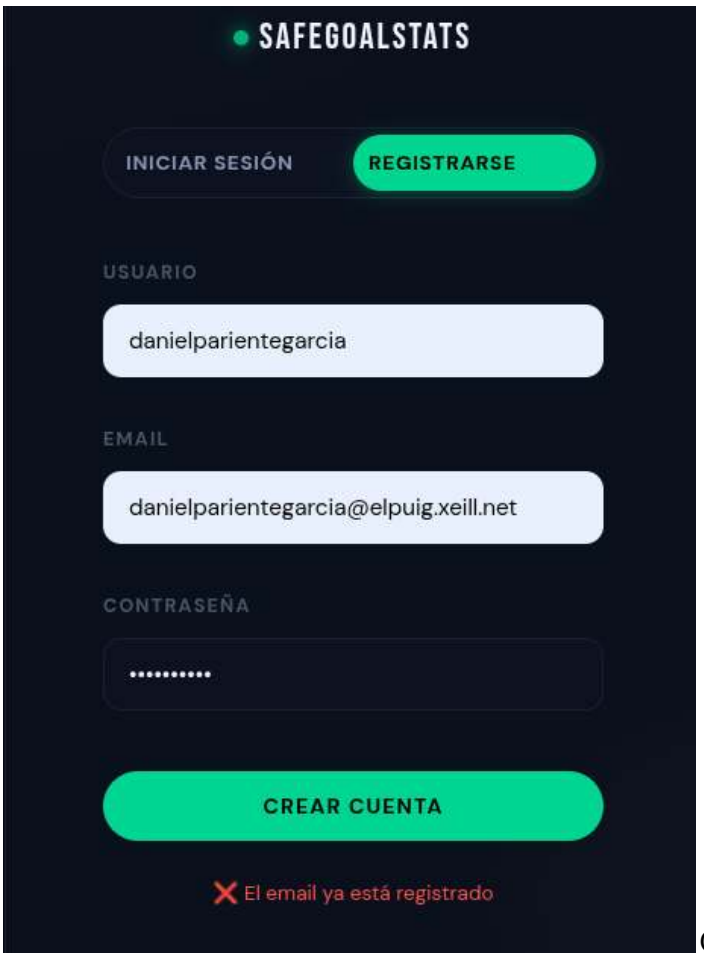
Aquí tenemos una comprobación


```
usuario@ubuntu-2404-desktop:/var/www/safegoalstats-api$ npm start

> safegoalstats-api@1.0.0 start
> node src/index.js

◇ injected env (7) from .env // tip: ◈ encrypted .env [www.dotenvx.com]
◇ injected env (0) from .env // tip: ◈ encrypted .env [www.dotenvx.com]
Servidor corriendo en http://localhost:3000
Conectado a MySQL correctamente
[LOGIN FAIL] email: danielparientegarcia@elpuig.xeill.net | IP: ::ffff:192.168.236.1 | Motivo: contraseña incorrecta | Hora: 202
6-05-07T14:19:01.329Z
[LOGIN OK] usuario: Dani | IP: ::ffff:192.168.236.1 | Hora: 2026-05-07T14:19:52.879Z
```

Y aparece perfectamente el cómo no nos ha dejado iniciar sesión y luego sí, y ambas aparecen registradas.



O por ejemplo si intentamos registrar un usuario que ya lo esta.

```
[LOGIN OK] usuario: Dani | IP: ::ffff:192.168.236.1 | Hora: 2026-05-07T14:23:02.865Z
[REGISTRO FAIL] email repetido: danielparientegarcia@elpuig.xeill.net | IP: ::ffff:192.168.236.1 | Hora: 2026-05-07T14:24:16.213
Z
```

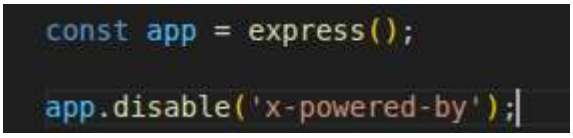
Y se registra perfectamente.

7. Implementación de HTTPS

Hemos mejorado la seguridad de la web implementando HTTPS, de forma que toda la información que viaja entre el usuario y SafeGoalStats ahora va cifrada. Esto es especialmente importante en apartados como el login, donde se envían credenciales sensibles.

También hemos hecho que la API ya no esté expuesta directamente al usuario, sino que ahora pasa antes por Apache, que actúa como intermediario. Con esto conseguimos una estructura más limpia, más segura y evitamos exponer innecesariamente el puerto interno del backend.

También hemos ocultado información técnica innecesaria del servidor.
Por defecto, Express avisa en las respuestas de que la web está hecha con esa tecnología.
Aunque parezca una tontería, eso da pistas a cualquiera que quiera buscar fallos concretos.
Eliminando esa información hacemos que el servidor exponga menos detalles internos.



Durante las pruebas detectamos que las estadísticas dejaban de cargar al entrar mediante HTTPS. Esto ocurría porque el frontend seguía intentando acceder a la API usando HTTP, algo que los navegadores modernos bloquean automáticamente por seguridad.

Para solucionarlo dejamos de acceder directamente al puerto interno del backend y empezamos a usar Apache como intermediario. De esta forma todas las peticiones pasan por HTTPS y la conexión se mantiene cifrada en todo momento.

También se eliminó el apartado de asistencias, ya que la API gratuita utilizada no ofrece datos reales de esa estadística y preferimos mostrar únicamente información auténtica y verificable.

11. Últimos problemas detectados durante el desarrollo

Durante la fase final del proyecto aparecieron varios problemas adicionales relacionados con la conexión entre la web, la API y el despliegue.

11.1 Error de API en estadísticas (token inválido)

Al probar la sección de estadísticas se detectó que la API devolvía el siguiente error:

{"message":"Your API token is invalid.","errorCode":400}

Esto ocurría porque el servicio externo de fútbol rechazaba la clave utilizada o no la reconocía correctamente.
Tras revisar las variables de entorno en Railway, se confirmó que el problema venía de la configuración del token de acceso a la API externa.

11.2 Problemas de rutas en frontend y backend

Se identificaron varios endpoints en el frontend que aún apuntaban a direcciones antiguas o no unificadas.
Esto provocaba que algunas peticiones no coincidieran correctamente con el servidor desplegado.

Se revisaron los archivos del frontend donde se realizaban llamadas fetch a la API para asegurar que todas utilizaban la misma URL base del backend desplegado.

11.3 Despliegue con Vercel (error de autenticación CLI)

Durante el despliegue del frontend en Vercel apareció un error relacionado con el token de acceso del CLI:

Error: The specified token is not valid. Use `vercel login`

Esto impidió completar el despliegue hasta que se volvió a iniciar sesión correctamente en la herramienta de Vercel.

11.4 Diferencias entre entorno local y producción

Se detectó que el comportamiento de la aplicación cambiaba entre la máquina virtual y dispositivos externos debido a diferencias en la configuración de acceso a la API.

El problema se debía a que en algunos casos se usaban URLs internas del servidor en lugar de la URL pública del backend.

---

12. Conclusión final del proyecto

SafeGoalStats
Daniel Pariente | Alex Rubio

El proyecto SafeGoalStats ha permitido desarrollar una aplicación web completa basada en arquitectura frontend y backend separados, conectados a una base de datos MySQL y a una API externa de fútbol.

A lo largo del desarrollo se han aprendido y aplicado conceptos importantes como:

- Gestión de autenticación con JWT
- Consumo de APIs externas
- Comunicación entre frontend y backend mediante fetch
- Configuración de variables de entorno
- Seguridad básica en aplicaciones web
- Despliegue en servicios cloud (Railway y Vercel)
- Resolución de problemas de rutas y conectividad

Además, durante las pruebas en local se demostró que la aplicación funciona correctamente en todos los dispositivos conectados a la misma red, permitiendo registro de usuarios, inicio de sesión y consulta de datos en tiempo real.

Como mejora futura, sería interesante desplegar la aplicación en un entorno totalmente público con dominio propio y configuración completa en producción, permitiendo acceso global sin depender de entornos locales o máquinas virtuales.