

UNIVERSIDAD MARIANO GÁLVEZ DE GUATEMALA  
SEDE BOCA DEL MONTE  
ING. SISTEMAS DE INFORMACIÓN Y CIENCIAS DE LA COMPUTACIÓN  
CURSO: DESARROLLO WEB



### PROYECTO FINAL

ANGEL ENRIQUE IBAÑEZ LINARES 7690-22-19119

BRYAN MANUEL PINEDA OROZCO 7690-16-8869

CESAR ALBERTO TECUN LEIVA 7690-22-11766

EDRAS FERNANDO TATUACA ALVARADO 7690-22-11542

JOSE DANIEL TOBAR REYES 7690-21-1325

PABLO ANTONIO ISPACHE ARRIAGA 7690-17-940

## # MANUAL TECNICO

### ## introducción

El presente documento describe los **\*\*aspectos técnicos del Sistema de Marcador de Baloncesto\*\***, una aplicación web desarrollada bajo arquitectura de microservicios. Su objetivo es brindar una **\*\*guía técnica a los desarrolladores y administradores del sistema, abarcando su estructura, componentes, base de datos, autenticación y mantenimiento general.\*\***

### ## Arquitectura del Sistema

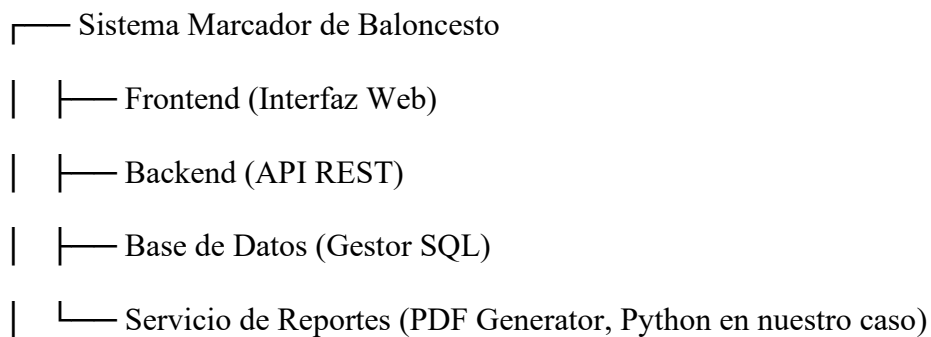
El sistema fue diseñado utilizando una arquitectura basada en microservicios. Cada módulo cumple una función específica e independiente, lo cual mejora la escalabilidad, el mantenimiento y la disponibilidad del sistema. A continuación, se detalla la estructura general:

**\*\*Frontend:\*\*** Interfaz web para la gestión del marcador, equipos, jugadores y reportes.

**\*\*Backend:\*\*** Servicio encargado de procesar las solicitudes, autenticar usuarios y administrar datos.

**\*\*Base de Datos:\*\*** Contiene la información relacionada con equipos, jugadores, partidos y estadísticas.

**\*\*Servicio de Reportes:\*\*** Genera documentos PDF con base en las consultas realizadas al sistema.



## ## Entorno de Desarrollo y Producción

El entorno de desarrollo del **Sistema de Marcador de Baloncesto** está compuesto por múltiples servicios que operan de manera independiente dentro de una arquitectura de microservicios, cada servicio se ejecuta de forma desacoplada para garantizar **portabilidad, escalabilidad y mantenibilidad**.

El sistema puede ejecutarse tanto en **modo desarrollo local** (usando Visual Studio Code y Node.js) como en **modo producción** (mediante orquestación con Docker Compose).

### ### Componentes principales del entorno

**Editor de código:** Visual Studio Code (entorno principal de desarrollo).

**Control de versiones:** Git y GitHub (repositorio centralizado).

**Contenedores y orquestación:** Docker y Docker Compose.

**Frontend:** Aplicación Angular ubicada en `/ui`.

**Backend (API):** Servicio principal dentro de `/api`.

**Reportes y PDF Renderer:** Servicios auxiliares que generan reportes en formato PDF.

**ETL:** Módulo encargado de la carga y transformación de datos.

**Base de datos:** Motor SQL SERVER alojado en el contenedor `/db`.

**Autenticación:** Módulo `auth-service` con soporte JWT y OAuth 2.0.

**Navegadores compatibles:** Google Chrome, Microsoft Edge, Mozilla Firefox.

## ## Estructura de Carpetas y Componentes

La estructura del proyecto **Desarrollo-Web-Proyecto-III** está organizada en módulos independientes que siguen **el principio de microservicios**.

Cada carpeta cumple un rol específico dentro del sistema, incluyendo **backend, frontend, reportería, ETL, base de datos y archivos de configuración general**.

## Carpetas principales del proyecto:

**\*\*api/\*\***

Contiene toda la **\*\*lógica del backend principal\*\*** (servicio de API).

Dentro de esta carpeta se encuentran los siguientes submódulos:

- **\*\*auth-service/\*\*** servicio de autenticación para manejo de usuarios y tokens JWT.
- **\*\*db/\*\*** scripts y configuraciones de conexión a la base de datos.
- **\*\*docs/\*\*** documentación técnica del backend.
- **\*\*etl/\*\*** módulo de extracción, transformación y carga de datos (ETL).
- **\*\*pdf-renderer/\*\*** servicio para la generación y renderizado de reportes en formato PDF.
- **\*\*reports/\*\*** módulo de reportería general.
  - **\*\*app/\*\*** contiene el código fuente del servicio de reportes.
  - **\*\*Dockerfile:\*\*** configuración del contenedor para el servicio.
  - **\*\*requirements.txt:\*\*** dependencias del servicio (bibliotecas necesarias).
- **\*\*scripts/\*\*** scripts utilitarios o de automatización.
- **\*\*users.json:\*\*** archivo con datos de usuarios, utilizado para pruebas o autenticación local.

**\*\*ui/\*\***

Carpeta del **\*\*frontend principal\*\*** desarrollado con **\*\*Angular.\*\***

Incluye todos los componentes visuales e interfaces del sistema.

Subcarpetas y archivos relevantes:

- **\*\*.angular/\*\*** configuración interna del entorno Angular.
- **\*\*node\_modules/\*\*** dependencias del frontend (se generan automáticamente).

- **public/** archivos estáticos, como imágenes, íconos o favicon.
- **src/** código fuente principal de la aplicación.
  - **app/** contiene los módulos y componentes Angular.
  - **assets/** almacena imágenes, estilos y recursos gráficos.
  - **environments/** contiene las configuraciones para desarrollo y producción.
  - **main.ts** punto de entrada de la aplicación Angular.
- **angular.json** configuración general del proyecto Angular.
- **package.json** dependencias y scripts de ejecución.
- **tsconfig.json**: configuración global de TypeScript.
- **nginx.conf** configuración del servidor NGINX para despliegue.
- **Dockerfile** configuración de imagen base para el frontend.

#### ## Archivos y configuración general del proyecto:

- **.env** variables de entorno globales utilizadas por los distintos servicios.
- **.gitignore** lista de archivos y carpetas que se excluyen del control de versiones.
- **docker-compose.yml** archivo que orquesta todos los servicios (API, frontend, reportes, base de datos, autenticación).
- **README.md** descripción general del proyecto.
- **Marcador-de-baloncesto.sln** archivo de solución principal.

#### ## Documentación y entregables:

- **Manual\_Usuario\_Marcador\_Baloncesto.pdf** documento del manual de usuario.
- **Manual\_usuario.md** versión en formato Markdown del manual de usuario.
- **Manual\_Tecnico\_Marcador\_Baloncesto.docx** manual técnico del sistema.
- **Tabla de Trabajo, Fase 2.png** imagen con avance o planificación del proyecto.

- **Tabla de Trabajo, Fase 2.md:** documento de avance asociado.

### ## Notas técnicas:

- La **estructura modular** facilita el mantenimiento y despliegue de cada componente por separado.
- Cada servicio puede ejecutarse **individualmente o dentro de un entorno Docker común.**
- El archivo **docker-compose.yml** centraliza la configuración para levantar todos los servicios.
- Los archivos sensibles como **.env** o **users.json** deben mantenerse fuera del repositorio público por razones de seguridad.

### ## Base de Datos

El sistema utiliza un **modelo de base de datos distribuido** bajo una arquitectura multimotor, donde cada microservicio gestiona su propio esquema de datos según su propósito.

Esto permite **mejorar el rendimiento, la independencia de componentes y la tolerancia a fallos.**

Cada base de datos se diseñó siguiendo principios de normalización y garantizando la integridad referencial entre entidades relacionadas.

### ## Proyecto Principal C# / SQL Server

El núcleo del sistema, desarrollado en **C#**, utiliza **SQL Server** como motor de base de datos.

Aquí se almacenan las entidades centrales del marcador, equipos, jugadores y partidos.

### Tablas principales:

- **Teams:** Información de los equipos registrados.
- **Players:** Datos de jugadores y su relación con los equipos.

- **Games:** Partidos programados o jugados, con estados y resultados.
- **GameEvents:** Eventos o acciones ocurridas durante un partido (faltas, puntos, tiempos).

Ejemplo de estructura SQL:

```
CREATE TABLE Teams (  
  TeamID INT PRIMARY KEY IDENTITY,  
  Name NVARCHAR(100) NOT NULL,  
  City NVARCHAR(100),  
  LogoUrl NVARCHAR(255),  
  CreatedAt DATETIME DEFAULT GETDATE()  
);
```

## Microservicio de Autenticación Node.js / MySQL / JWT

El servicio de autenticación maneja el registro, login y control de accesos de los usuarios.

Utiliza **MySQL** por su eficiencia en operaciones concurrentes y bajo consumo de recursos.

Este microservicio trabaja con **JWT (JSON Web Tokens)** y **OAuth 2.0** para la validación de credenciales.

Tablas principales:

- **Users:** Información básica de usuarios (nombre, correo, contraseña encriptada, rol).
- **Sessions:** Tokens activos, fechas de expiración y estado de autenticación.
- **Roles:** Niveles de acceso (Administrador, Árbitro, Operador, etc.).

Ejemplo de estructura MySQL:

```
1 CREATE TABLE Users (  
2   UserID INT AUTO_INCREMENT PRIMARY KEY,  
3   Username VARCHAR(100) NOT NULL,  
4   Email VARCHAR(150) UNIQUE NOT NULL,  
5   PasswordHash VARCHAR(255) NOT NULL,  
6   Role VARCHAR(50),  
7   CreatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
8 );  
9
```

## Microservicio de Reportes, Python / PostgreSQL

El microservicio de **reportería** se encarga de la generación de reportes PDF y el almacenamiento temporal de consultas agregadas.

Utiliza **PostgreSQL** por su robustez y compatibilidad con estructuras JSON y operaciones analíticas.

Tablas principales:

- **ReportRequests:** Registra cada solicitud de generación de reportes (usuario, tipo, fecha, estado).
- **GeneratedReports:** Almacena metadatos de reportes generados, rutas de acceso y fechas de creación.
- **Logs:** Guarda los registros de ejecución del servicio para auditoría y depuración.

Ejemplo de estructura PostgreSQL:



```
CREATE TABLE ReportRequests (
  RequestID SERIAL PRIMARY KEY,
  UserID INT NOT NULL,
  ReportType VARCHAR(100) NOT NULL,
  RequestDate TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  Status VARCHAR(50) DEFAULT 'PENDING'
);
```

### ## Despliegue y Ejecución del Sistema

El Sistema de **Marcador de Baloncesto** está diseñado para ejecutarse mediante contenedores **Docker**, lo que permite una configuración uniforme del entorno y facilita el despliegue tanto en desarrollo como en producción.

Cada microservicio **(API, frontend, autenticación, reportes, PDF-renderer y ETL)** se levanta automáticamente mediante el archivo **docker-compose.yml**, el cual define la orquestación completa del sistema.

### ## Requisitos previos:

Antes de iniciar el despliegue, asegúrese de contar con los siguientes componentes instalados:

- **Docker Engine** (versión 24 o superior).
- **Docker Compose** (integrado en Docker Desktop).
- **Git** (para clonar el repositorio o sincronizar versiones).
- **Visual Studio Code** u otro editor de código compatible.

### ## Procedimiento de ejecución:

1. Abra una terminal en la **carpeta raíz** del proyecto:

Desarrollo-Web-Proyecto-III-y-Final/

2. Ejecute el siguiente comando para construir e iniciar todos los servicios definidos:

```
**docker-compose --profile all up --build**
```

3. Docker descargará las imágenes necesarias (si no existen localmente) y construirá los contenedores definidos en el archivo **\*\*docker-compose.yml\*\***

4. Una vez completada la construcción, los servicios quedarán activos en los siguientes puertos predeterminados:

- **\*\*Backend\*\*** (API principal): <http://localhost:8080>
- **\*\*Frontend\*\*** (Angular): <http://localhost:4200>
- **\*\*Reportes:\*\*** <http://localhost:5000>
- **\*\*PDF Renderer:\*\*** <http://localhost:5050>
- **\*\*Auth Service\*\***: <http://localhost:7070>
- **\*\*Base de datos\*\***: Puerto interno 3306 (MySQL) y 5432 (PostgreSQL para reportes).

## Verificación del entorno:

- Acceda al **\*\*frontend\*\*** en el navegador para validar el correcto funcionamiento de la interfaz gráfica.
- Desde el **\*\*panel principal,\*\*** pruebe las opciones de autenticación, gestión de equipos, jugadores y reportería.
- Verifique los registros en la terminal de **\*\*Docker\*\*** para confirmar que todos los servicios se levantaron correctamente.

## Finalización de los servicios:

Para detener la ejecución de todos los contenedores, utilice el siguiente comando:

```
**docker-compose down**
```

### ## Notas técnicas:

- **El parámetro --profile all** permite levantar todos los microservicios definidos en el entorno.
- **El parámetro --build** asegura que se recompilen las imágenes antes de iniciar los contenedores, lo cual es útil cuando se han realizado cambios en el código fuente.
- En entornos de producción, se recomienda utilizar el mismo comando acompañado de variables **dotenv (.env)** con credenciales y configuraciones seguras.

### ## Resultado esperado:

Una vez levantado el entorno, el sistema quedará disponible en la red local con todos los módulos ejecutándose de manera sincronizada bajo la misma arquitectura de **microservicios**.

#### Seguridad y Autenticación.

El sistema implementa un modelo de **seguridad híbrido** basado en estándares modernos de autenticación y control de acceso, con el objetivo de garantizar la **integridad, confidencialidad y disponibilidad** de la información.

Durante la **Fase 3 del desarrollo**, se integraron mecanismos de autenticación tanto locales como externos, permitiendo a los usuarios y administradores acceder de forma segura a las diferentes áreas del sistema.

### ## Autenticación mediante JWT (JSON Web Token):

El sistema utiliza **tokens JWT** para el manejo de sesiones seguras en el módulo principal.

Cada usuario autenticado recibe un **token firmado** que contiene su información de identificación, rol y tiempo de expiración.

Este **token** se envía junto con cada solicitud al servidor para validar el acceso a los recursos protegidos.

Características principales:

- **\*\*Emisión\*\*** de tokens únicos por sesión.
- **\*\*Validación\*\*** automática en cada solicitud al API principal.
- **\*\*Expiración\*\*** controlada del token para evitar sesiones prolongadas.
- **\*\*Protección\*\*** frente a manipulación mediante firma criptográfica.

## Autenticación mediante OAuth 2.0 (Inicio de sesión con GitHub):

En esta fase se implementó la autenticación externa utilizando el protocolo **\*\*OAuth 2.0,\*\*** permitiendo a los usuarios acceder al sistema empleando sus credenciales de GitHub.

Este método evita el almacenamiento local de contraseñas y mejora la seguridad general, delegando la verificación de identidad a un proveedor confiable.

Características destacadas:

- **\*\*Inicio de sesión rápido\*\*** y seguro con cuentas de GitHub.
- **\*\*Obtención de permisos limitados\*\*** (solo lectura de perfil básico).
- **\*\*Asociación automática\*\*** con el rol correspondiente en el sistema.
- **\*\*Integración con el módulo\*\*** de auditoría para registrar accesos.

## Acceso SSH exclusivo para ingenieros administradores:

El sistema contempla **\*\*un mecanismo adicional\*\*** de acceso seguro mediante **\*\*llave SSH\*\*** para los ingenieros encargados del mantenimiento del servidor y la infraestructura del sistema.

Este acceso está limitado a usuarios con **\*\*privilegios de administración\*\*** y se utiliza únicamente para operaciones técnicas como actualizaciones, respaldos o monitoreo de contenedores.

Llave registrada:

**ed25519:**

**sshed25519AAAAC3NzaC1lZDI1NTE5AAAAIHx1yR2zNcjAFGdWn4fvuzqak1n6shVOJ**

**Edv/DfXcWSMelgust**

**## Políticas de seguridad complementarias:**

- Todos los servicios se **\*\*comunican\*\*** mediante canales cifrados.
- Los tokens y credenciales **\*\*no se almacenan\*\*** en texto plano.
- Los accesos de **\*\*OAuth y SSH\*\*** son registrados en bitácoras internas.
- Se **\*\*implementa control de roles y permisos\*\*** jerárquicos (Administrador y Viewer)

**## Seguridad OWASP**

Código OWASP	Riesgo Mitigado	Controles Implementados
A01: Broken Access Control	Acceso indebido a funciones o recursos	Validación estricta de roles (ADMIN / USER / VIEWER), middleware en API, rutas protegidas y control en UI
A02: Cryptographic Failures	Exposición de datos sensibles	JWT firmado, HTTPS, gestión de secretos en variables de entorno, no almacenamiento de claves en código
A03: Injection	Inyección SQL/JS en formularios y consultas	Sanitización de campos, uso de ORM seguro, validación en backend y frontend
A04: Insecure Design	Falta de barreras de seguridad en diseño	Arquitectura por microservicios, aislamiento de servicios, control de sesiones y permisos
A05: Security Misconfiguration	Configuración insegura en servidores y contenedores	Hardening en Docker y servicios, CORS configurado, puertos restringidos
A06: Outdated Components	Uso de librerías vulnerables	Auditorías periódicas ( <code>npm audit</code> , dependencias .NET, Python), actualización y eliminación de paquetes obsoletos
A07: Identification & Authentication Failures	Vulnerabilidades en autenticación y sesiones	JWT expirables, OAuth2 (GitHub), protección contra fuerza bruta
A08: Software Integrity Failures	Manipulación de código o librerías	Verificación de integridad, control de versiones, validación de dependencias externas
A09: Logging & Monitoring Failures	Falta de registros que detecten ataques	Logs centralizados, auditoría básica de accesos y eventos del sistema

Código OWASP	Riesgo Mitigado	Controles Implementados
A10: SSRF (Server-Side Request Forgery)	Solicitudes internas no autorizadas	Validación de URLs, bloqueo de accesos internos no permitidos

## ## Generación de Reportes

El sistema cuenta con un **módulo especializado** en la generación de reportes en formato PDF, desarrollado en Python con conexión a una base de datos PostgreSQL.

Este módulo se comunica con el **backend principal**, implementado en C# y conectado a SQL Server, a través de una **API REST** que provee los datos necesarios para la creación de los reportes.

La **interfaz gráfica**, desarrollada en Angular, permite al usuario generar, filtrar y descargar reportes en tiempo real desde un entorno web moderno y seguro.

La **reportería** ofrece una visión consolidada de la información almacenada, abarcando aspectos como equipos, jugadores, historial de partidos, roster y estadísticas individuales.

Cada documento **PDF** mantiene la identidad visual del sistema y se genera bajo un formato estructurado y homogéneo.

### ## RF-REP-01: **Reporte de Equipos**

Genera un PDF con la información general de los equipos registrados.

El usuario puede buscar por nombre o filtrar por ciudad antes de presionar el botón “Descargar PDF de Equipos”, que genera el documento automáticamente.

### ##RF-REP-02: **Reporte de Jugadores por Equipo**

Permite generar un listado de jugadores asociados a un equipo.

Incluye información detallada como nombre, posición, edad, estatura y nacionalidad.

El usuario selecciona el equipo y descarga el reporte mediante “Descargar PDF de Jugadores”.

#### ## RF-REP-03: \*\*Historial de Partidos\*\*

Genera un informe con todos los partidos jugados o programados.

Los resultados pueden filtrarse por estado (Finalizado, En curso o Cancelado).

El botón “Descargar PDF de Partidos” genera el documento con los datos filtrados.

#### ## RF-REP-04: \*\*Roster por Partido\*\*

Muestra la alineación de jugadores para un partido específico.

El usuario selecciona el partido o ingresa manualmente el ID del partido, y descarga el reporte mediante “Descargar PDF de Roster”.

#### ## RF-REP-05: \*\*Estadísticas por Jugador\*\*

Permite generar un reporte individual por jugador, con información estadística de su rendimiento en los partidos.

El usuario selecciona el equipo, el jugador y opcionalmente ingresa el ID del jugador para generar el documento mediante “Descargar PDF de Estadísticas”.

#### ## Funcionalidades adicionales:

- **\*\*Botón Actualizar:\*\*** Recarga los datos visualizados en pantalla.
- **\*\*Modo Claro/Oscuro:\*\*** Permite alternar el esquema de color de la interfaz Angular.
- **\*\*Botón Cerrar Sesión:\*\*** Cierra la sesión activa de forma segura.

#### ## Aspectos técnicos del módulo:

- **La reportería está desarrollada en Python,** utilizando librerías para procesamiento de datos y generación de documentos PDF.
- El **backend principal en C# (SQL Server)** expone los datos mediante API REST.
- El **frontend Angular** envía las solicitudes y gestiona la descarga de archivos PDF.
- Las **peticiones se registran en la base de datos PostgreSQL** del microservicio de reportes.
- Los documentos se generan en tiempo real con formato **PDF/A**, garantizando compatibilidad y persistencia.
- El sistema **maneja errores controlados en caso de consultas vacías o interrupciones entre servicios.**
- Todos los **reportes incluyen encabezado, logotipo y fecha de generación** bajo un formato uniforme.

### ## Mantenimiento y Actualizaciones

El mantenimiento del sistema de **Marcador de Baloncesto** se centra en la estabilidad operativa, la integridad de los datos y la continuidad del servicio en los diferentes entornos desplegados mediante contenedores Docker.

Cada componente **frontend, backend, microservicios y base de datos** requiere seguimiento individual y actualizaciones controladas para evitar interrupciones.

### ### Copias de seguridad y recuperación de datos

- Se recomienda **generar backups automáticos de las bases de datos SQL Server (módulo principal) y PostgreSQL (microservicio de reportes) al menos una vez por semana.**



- Las copias **\*\*deben almacenarse en un entorno externo o volumen montado dentro del contenedor Docker correspondiente.\*\***
- En caso de falla del sistema, **\*\*los datos pueden restaurarse desde los respaldos utilizando los scripts SQL o dump generados automáticamente.\*\***

## ## Actualización de componentes y librerías

- **\*\*El entorno de Angular debe mantenerse actualizado ejecutando periódicamente los comandos npm update y npm audit fix para corregir vulnerabilidades.\*\***
- **\*\*En el backend C#, se recomienda revisar las dependencias del proyecto en el archivo .csproj y aplicar actualizaciones mediante dotnet restore y dotnet build.\*\***
- **\*\*Para el microservicio de reportes en Python, se deben revisar y actualizar los paquetes definidos en requirements.txt mediante el comando pip install -r requirements.txt --upgrade.\*\***

## ## Supervisión del rendimiento y logs

- Los registros de cada servicio **\*\* (API, UI y reportes) deben revisarse regularmente\*\*** mediante los comandos docker logs <nombre\_del\_contenedor> o mediante herramientas de monitoreo integradas.
- Es recomendable **\*\*implementar métricas básicas de rendimiento, como tiempo de respuesta de API, tamaño de los reportes generados y consumo de memoria de los contenedores.\*\***
- En caso de **\*\*detectar errores recurrentes, los logs deben exportarse para su análisis y depuración.\*\***

## ## Control de versiones y despliegue

- Todas las actualizaciones deben **\*\*gestionarse bajo control de versiones en el repositorio oficial de GitHub\*\*** (<https://github.com/Pispache/Desarrollo-Web-Proyecto-III-y-Final>).
- Se sugiere utilizar **\*\*ramas dedicadas (develop, testing, production) para aplicar cambios sin afectar el entorno estable.\*\***
- Antes de desplegar actualizaciones en producción, **\*\*probar los cambios localmente mediante Docker Compose y revisar la correcta ejecución de los servicios.\*\***

### ## Verificación post-actualización

- **\*\*Confirmar el acceso a los módulos de login, control de partidos y reportería PDF tras cada actualización.\*\***
- **\*\*Verificar la comunicación entre los microservicios (API C# ↔ Python ↔ Angular).\*\***
- **\*\*Validar la integridad de las bases de datos y la correcta regeneración de los tokens JWT y accesos OAuth tras reiniciar los contenedores.\*\***

### ## Soporte Técnico y Créditos

El soporte técnico del sistema Marcador de Baloncesto está a cargo del equipo de desarrollo responsable del diseño, implementación y mantenimiento de los módulos principales:

**\*\*Frontend (Angular), Backend (C# / SQL Server) y Microservicio de Reportes (Python / PostgreSQL).\*\***

Para consultas, reportes de errores o solicitudes de mantenimiento, los usuarios pueden comunicarse directamente con cualquiera de los siguientes desarrolladores mediante sus correos institucionales:

### ## Soporte Técnico:

- **\*\*Ángel Enrique Ibañez Linares\*\*** – [aibanezl@miumg.edu.gt](mailto:aibanezl@miumg.edu.gt)

- **Bryan Manuel Pineda Orozco** – bpinedao@miumg.edu.gt
- **César Alberto Tecún Leiva** – ctecunl1@miumg.edu.gt
- **Edras Fernando Tatuaca Alvarado** – etatuacaa@miumg.edu.gt
- **José Daniel Tobar Reyes** – jtobarr5@miumg.edu.gt
- **Pablo Antonio Ispache Arriaga** – pispachea@miumg.edu.gt

## Repositorio oficial del proyecto:

**<https://github.com/Pispache/Desarrollo-Web-Proyecto-III-y-Final>**

**Universidad Mariano Gálvez de Guatemala**

**Facultad de Ingeniería en Sistemas de la Información**

**Año: 2025**