[DOT & DOT] Diseño del Backend

Idea

El backend desarrollado actúa como árbitro central de las comunicaciones del ecosistema de aplicaciones. Esta aplicación se encarga de:

Responsabilidades externas:

1. Registrar y autorizar usuarios mediante sistema de verificación por email.
2. Implementar control de acceso basado en roles (ROLE_ADMIN, ROLE_REFEREE, ROLE_USER).
3. Gestionar el almacenamiento, modificación y visualización de partidos en tiempo real.
4. Validar dispositivos y sesiones mediante tokens JWT con refresh tokens.
5. Proporcionar endpoints públicos para consulta de información sin autenticación.
6. Comunicación en tiempo real mediante WebSockets (STOMP) para actualización de partidos.

Responsabilidades internas:

1. Persistencia en base de datos MySQL con JPA/Hibernate.
2. Verificación de integridad referencial y validaciones a nivel de entidad.
3. Control de accesos mediante Spring Security con @PreAuthorize.
4. Gestión de imágenes con almacenamiento en servidor.
5. Servicio de correo electrónico para verificaciones y notificaciones.
6. Broadcasting de eventos de partidos mediante WebSocket Service.

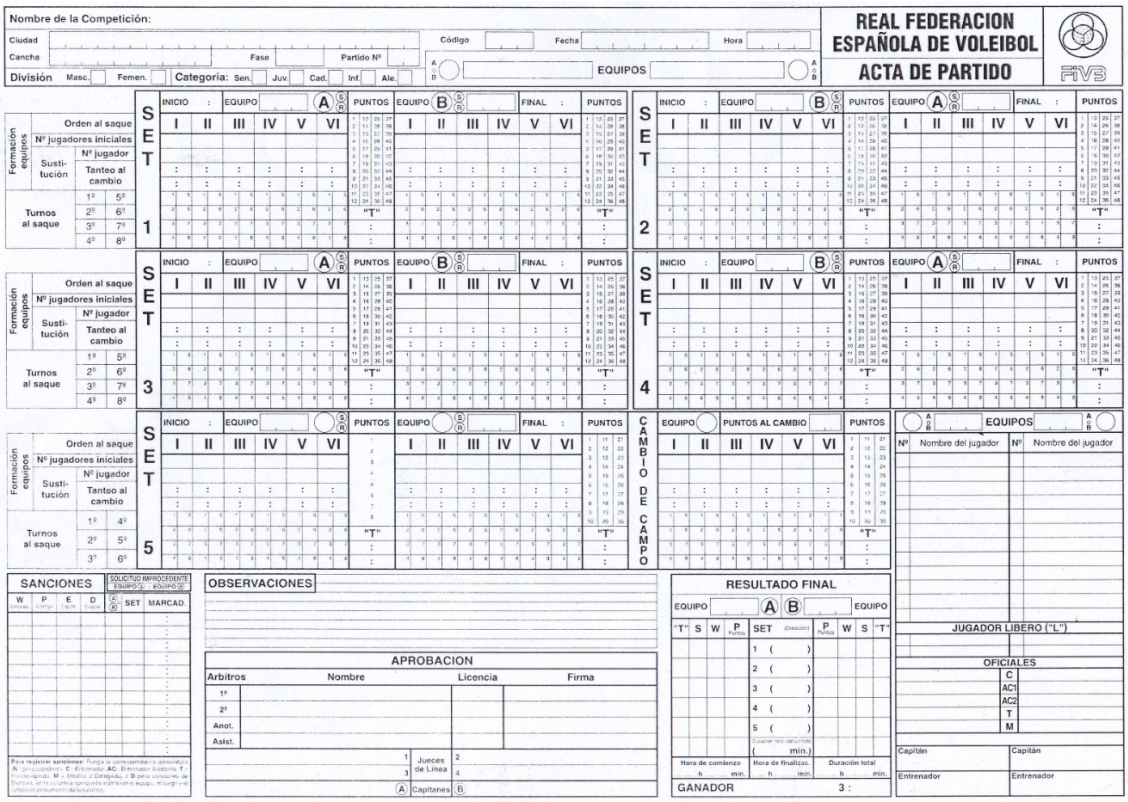
Estructura del Sistema

El backend se organiza en los siguientes módulos principales:

1. **Admin** - Gestión completa de entidades (CRUD) con rol ROLE_ADMIN
2. **Referee** - Endpoints específicos para árbitros durante partidos
3. **Open/Public** - Información pública accesible sin autenticación
4. **Auth** - Autenticación, registro, verificación y gestión de roles
5. **WebSocket** - Comunicación bidireccional en tiempo real
6. **Images** - Servicio de almacenamiento y recuperación de imágenes
7. **Mail** - Envío de correos de verificación y contacto

Modelo de Datos: Partidos

El sistema implementa una estructura completa que digitaliza el acta real de voleibol:



Entidad Game (Partido)

```
@Entity
public class Game {
    private long id;
    private String uniqueCode;                    // Código único del partido
    private GameDetails details;                  // Detalles (fecha, ciudad,
competición)
    private GameInitialSituation initialSituation;  // Situación inicial (equipos,
saque)
    private GameRefereeTeam refereeTeam; // Equipo arbitral asignado
    private GameObservations observation; // Observaciones del partido
    private List<GameSanctions> sanctionsList;  // Lista de sanciones
    private List<GameSet> sets;           // Sets del partido (ordenados)
    private GameResult result;            // Resultado final
    private boolean playing;              // Estado: en juego
    private boolean finished;             // Estado: finalizado
    private int relevance;                // Relevancia (partido destacado)
    private League league;                // Liga a la que pertenece
}
```

GameDetails (Detalles del Partido)

```
@Entity
public class GameDetails {
    private long id;
    private Game game;
    private Category category;      // INFANTIL, CADETE, JUVENIL, SENIOR
    private Division division;      // PRIMERA, SEGUNDA, TERCERA
    private Competition competition; // Competición (Liga regular, Copa, etc.)
    private City city;              // Ciudad donde se juega
    private Date date;              // Fecha del partido
    private Date timeStart;         // Hora de inicio real
}
```

GameSet (Set del Partido)

```
@Entity
public class GameSet {
    private long id;
    private String uniqueCode;      // Referencia al partido
    private int setNumber;          // Número del set (1-5)
    private int pointsLocal;        // Puntos equipo local
    private int pointsVisit;        // Puntos equipo visitante
    private Date timeStart;         // Hora de inicio del set
    private Date timeEnd;           // Hora de fin del set
    private String localAlignment;  // Alineación equipo local (JSON)
    private String visitAlignment;  // Alineación equipo visitante (JSON)
}
```

GameInitialSituation (Situación Inicial)

```
@Entity
public class GameInitialSituation {
    private long id;
    private Game game;
    private Team localTeam;         // Equipo local
    private Team visitTeam;         // Equipo visitante
    private Team servingTeam;       // Equipo que saca
    private Team receivingTeam;     // Equipo que recibe
    private Team leftTeam;          // Equipo situado a la izquierda
    private Team rightTeam;         // Equipo situado a la derecha
}
```

GameSanctions (Sanciones del Partido)

```
@Entity
public class GameSanctions {
    private long id;
    private Game game;
    private SanctionType type;      // YELLOW, RED, EXPULSION, DISQUALIFICATION
    private Long teamId;            // Equipo sancionado
    private String marcador;        // Marcador en el momento de la sanción
    private Date timestamp;         // Momento de la sanción
}
```

GameResult (Resultado Final)

```
@Entity
public class GameResult {
    private long id;
    private Game game;
    private Team winner;            // Equipo ganador
    private Team loser;             // Equipo perdedor
    private Date timeStart;         // Hora inicio del partido
    private Date timeEnd;           // Hora fin del partido
    private Long duration;          // Duración en milisegundos
    private int setsWonByLocal;     // Sets ganados por local
    private int setsWonByVisit;     // Sets ganados por visitante
}
```

Modelo de Datos: Personas y Equipos

Person (Persona - Clase Base)

```
@Entity
public class Person {
    private long id;
    private String dni;             // DNI único (12 caracteres)
    private String name;            // Nombre (20 caracteres)
    private String surnames;        // Apellidos (60 caracteres)
    private Date birthDate;         // Fecha de nacimiento
    private String address;         // Dirección (100 caracteres)
    private String phone;           // Teléfono (25 caracteres)
    private String email;           // Email (70 caracteres)
    private Boolean dniVerified;    // DNI verificado
    private Boolean tutored;        // Es tutelado (menor)
    private Person tutor;           // Tutor (si es menor)

    // Relaciones especializadas
    private Coach coach;            // OneToOne si es entrenador
    private Player player;          // OneToOne si es jugador
    private Referee referee;        // OneToOne si es árbitro
}
```

Player (Jugador)

```
@Entity
public class Player {
    private long id;
    private String dni;             // FK a Person
    private Integer shirtNumber;    // Número de camiseta
    private Team team;              // Equipo al que pertenece
    private Category category;      // Categoría del jugador
}
```

Coach (Entrenador)

```
@Entity
public class Coach {
    private long id;
    private String dni;             // FK a Person
    private String licenseNumber;   // Número de licencia
    private String licenseLevel;    // Nivel de licencia
    private Team team;              // Equipo al que pertenece
}
```

Referee (Árbitro)

```
@Entity
public class Referee {
    private long id;
    private String dni;             // FK a Person
    private String licenseNumber;   // Número de licencia
    private String licenseLevel;    // Nivel de licencia
    private City city;              // Ciudad de residencia
}
```

Team (Equipo)

```
@Entity
public class Team {
    private long id;
    private String name;            // Nombre del equipo
    private Person dniCaptain;      // Capitán del equipo
    private Club idClub;            // Club al que pertenece
    private Category category;      // Categoría del equipo
}
```

Club

```
@Entity
public class Club {
    private long id;
    private String name;            // Nombre del club
    private City city;              // Ciudad del club
}
```

Modelo de Datos: Definiciones y Lugares

City (Ciudad)

```
@Entity
public class City {
    private long id;
    private String name;            // Nombre de la ciudad
    private String region;          // Región/Provincia
    private String cpRange;         // Rango de códigos postales
}
```

Gameplace (Cancha/Pabellón)

```
@Entity
public class Gameplace {
    private long id;
    private String name;            // Nombre del pabellón
    private City city;              // Ciudad donde está ubicado
    private String address;         // Dirección completa
    private Integer courtsNumber;   // Número de pistas disponibles
}
```

League (Liga)

```
@Entity
public class League {
    private long id;
    private String name;            // Nombre de la liga
    private Competition competition; // Competición a la que pertenece
    private Category category;      // Categoría
    private Division division;      // División
    private Season season;          // Temporada
}
```

Competition (Competición)

```
@Entity
public class Competition {
    private long id;
    private String name;            // Nombre de la competición
    private String description;     // Descripción
    private Season season;          // Temporada
}
```

Season (Temporada)

```
@Entity
public class Season {
    private long id;
    private String name;            // Ej: "National 2024-2025"
    private Date startDate;         // Fecha de inicio
    private Date endDate;           // Fecha de finalización
    private Boolean active;         // Temporada activa
}
```

Enumeraciones Implementadas

```
public enum Category {
    PRE_BENJAMIN(0, "Pre-Benjamin", 6, 7),
    BENJAMIN(1, "Benjamin", 8, 9),
    ALEVIN(2, "Alevin", 10, 11),
    INFANTILE(3, "Infantile", 12, 13),
    CADET(4, "Cadet", 14, 15),
    YOUTH(5, "Youth", 16, 17),
    JUNIOR(6, "Junior", 18, 19),
    ABSOLUTE(7, "Absolute", 20, 39),
    SENIOR(8, "Senior", 40, 99);
}

public enum Division {
    MIXED('X', "Mixed"),
    FEMININE('F', "Feminine"),
    MASCULINE('M', "Masculine");
}

public enum SanctionType {
    INDIVIDUAL('I', "Individual"),
    COACH('C', "Coach"),
    ASSISTANT_COACH('A', "Assistant Coach"),
    SUPPORT('T', "Support"),
    IMPROPER('S', "Improper solicitude"),
    DELAY('D', "Delay");
}
```

Sistema de Autenticación y Autorización

Spring Security + JWT

El sistema utiliza Spring Security con tokens JWT (JSON Web Tokens) para autenticación y autorización.

User (Usuario del Sistema)

```
@Entity
@Table(name = "users")
public class User {
    private Long id;
    private String username;        // Nombre de usuario único
    private String email;           // Email único
    private String password;        // Contraseña cifrada con BCrypt
    private Boolean enabled;        // Usuario habilitado
    private Boolean verified;       // Email verificado
    private Set<Role> roles;        // Roles asignados
    private Instant createdAt;      // Fecha de creación
    private Instant updatedAt;      // Última actualización
}
```

Role (Roles del Sistema)

```
@Entity
@Table(name = "roles")
public class Role {
    private Long id;
    private String name;            // Nombre del rol
    private Instant createdAt;
    private Instant updatedAt;
}
```

Roles Implementados

- **ROLE_ADMIN**: Control total sobre la plataforma. CRUD completo de todas las entidades.
- **ROLE_REFEREE**: Gestión de partidos asignados, registro de puntos y sanciones en tiempo real.
- **ROLE_USER**: Acceso a perfil personal, consulta de información pública, favoritos.

Endpoints de Autenticación

AuthController (`/auth`):

1. **POST /auth/register**
    - Registro de nuevos usuarios
    - Envío de email de verificación
    - Respuesta: {status: 201, message: "User created"}
2. **POST /auth/login**
    - Autenticación con username/email y password
    - Respuesta: JWT token + refresh token
    - Formato: {status: 200, message: {token, refreshToken, username, email, roles}}
3. **GET /auth/verify?token={token}**
    - Verificación de email mediante token
    - Activa la cuenta del usuario
4. **GET /auth/forgot-password?email={email}**
    - Solicitud de reset de contraseña
    - Envío de email con token de recuperación
5. **POST /auth/reset-password**
    - Reseteo de contraseña con token
    - Body: {token, newPassword}
6. **GET /auth/validate**
    - Validación de sesión activa mediante JWT
    - Requiere: Authorization: Bearer {token}
7. **GET /auth/role/{role}**
    - Verificación de rol específico del usuario autenticado
    - Requiere: Authorization: Bearer {token}

PermissionController (`/permissions`)

Gestión de permisos y roles (ROLE_ADMIN):

1. **GET /permissions/roles** - Listar todos los roles
2. **POST /permissions/roles** - Crear nuevo rol
3. **PUT /permissions/users/{userId}/roles** - Asignar/modificar roles de usuario

Seguridad Implementada

- **BCryptPasswordEncoder**: Cifrado de contraseñas con algoritmo BCrypt
- **JWT**: Tokens firmados con secret key, expiración configurable
- **Refresh Tokens**: Renovación de sesión sin re-autenticación
- **CORS**: Configuración permisiva con `@CrossOrigin("*")` en controladores
- **@PreAuthorize**: Anotaciones en métodos para control de acceso por rol

API REST: Módulo Admin

El módulo admin proporciona endpoints CRUD completos para todas las entidades. Requiere **ROLE_ADMIN**.

Estructura General de Controladores

Todos los controladores admin siguen el mismo patrón CRUD estándar:

- **GET** `/admin/{resource}` - Listar todos los elementos
- **GET** `/admin/{resource}/{id}` - Obtener elemento por ID
- **POST** `/admin/{resource}` - Crear nuevo elemento
- **PUT** `/admin/{resource}/{id}` - Actualizar elemento
- **DELETE** `/admin/{resource}/{id}` - Eliminar elemento

Controladores Implementados

Gestión de Personas

- **PersonController** (`/admin/persons`) - Personas del sistema
- **PlayerController** (`/admin/players`) - Jugadores
- **CoachController** (`/admin/coaches`) - Entrenadores
- **RefereeController** (`/admin/referees`) - Árbitros

Gestión de Clubes y Equipos

- **ClubController** (`/admin/clubs`) - Clubes
- **TeamController** (`/admin/teams`) - Equipos

Gestión de Ubicaciones

- **CitiesController** (`/admin/cities`) - Ciudades
- **GameplacesController** (`/admin/gameplaces`) - Pabellones/Canchas

Gestión de Competiciones

- **SeasonController** (`/admin/seasons`) - Temporadas
- **CompetitionController** (`/admin/competitions`) - Competiciones
- **LeagueController** (`/admin/leagues`) - Ligas

Gestión de Partidos

- **GameController** (`/admin/games`) - Partidos

Endpoints Especializados del GameController

Además de los endpoints CRUD estándar, GameController incluye:

GET /admin/games/code/{uniqueCode}

Obtiene un partido por su código único en lugar de por ID numérico.

- **Uso**: Búsqueda rápida cuando se conoce el código del partido (ej: "1A_PRE_AAA000")
- **Response**: Objeto Game completo con todas sus relaciones

GET /admin/games/refereeable

Lista partidos disponibles para asignar árbitro.

- **Filtros aplicados**: partidos no finalizados, no en curso, sin árbitro asignado
- **Uso**: Vista para asignación de árbitros a partidos pendientes

API REST: Módulo Referee

El módulo referee proporciona endpoints específicos para árbitros durante la gestión de partidos. Requiere **ROLE_REFEREE**.

RefereeGamesController (`/referee/games`)

GET /referee/games/{refereeId}

Obtiene todos los partidos asignados a un árbitro específico.

- **Auth**: ROLE_REFEREE
- **Response**: Lista de partidos asignados

RefereeMatchController (`/referee/matches`)

GET /referee/matches/{uniqueCode}

Obtiene los detalles completos de un partido por código único.

- **Auth**: ROLE_REFEREE
- **Response**: Objeto Game con todas sus relaciones

PUT /referee/matches/{uniqueCode}/start

Inicia un partido, creando el primer set automáticamente.

- **Auth**: ROLE_REFEREE
- **Lógica**:
  - Valida que el partido no esté ya en curso
  - Crea GameSet #1 con puntos en 0
  - Marca game.playing = true
  - Broadcast WebSocket: "STARTED"
- **Response**: Game actualizado

POST /referee/matches/{uniqueCode}/sets/{setId}/points

Actualiza los puntos de un set durante el partido.

- **Auth**: ROLE_REFEREE
- **Body**: `{team: "local"|"visit", points: Integer}`
- **Lógica**:
  - Incrementa puntos del equipo especificado
  - Valida reglas de finalización de set (25 puntos, diferencia de 2)
  - Finaliza set automáticamente si cumple condiciones
  - Crea siguiente set si es necesario
  - Finaliza partido automáticamente si un equipo gana 3 sets
  - Broadcast WebSocket: "POINT_UPDATE"
- **Response**: Set actualizado

POST /referee/matches/{uniqueCode}/sanctions

Registra una sanción durante el partido.

- **Auth**: ROLE_REFEREE
- **Body**: `{type: SanctionType, teamId: Long, marcador: String}`
- **Lógica**:
  - Crea GameSanctions con timestamp actual
  - Asocia al partido y equipo correspondiente
  - Broadcast WebSocket: "SANCTION_ADDED"
- **Response**: Sanción creada

PUT /referee/matches/{uniqueCode}/finish

Finaliza un partido manualmente.

- **Auth**: ROLE_REFEREE
- **Lógica**:
  - Finaliza el set activo
  - Calcula resultado final (ganador, perdedor, duración)
  - Crea GameResult con estadísticas
  - Marca game.finished = true
  - Broadcast WebSocket: "FINISHED"
- **Response**: Game finalizado con resultado

API REST: Módulo Open/Public

El módulo open proporciona endpoints públicos accesibles sin autenticación para consulta de información.

OpenGameController (`/open/games`)

GET /open/games

Lista todos los partidos disponibles para el público.

- **Auth**: No requiere
- **Response**: Lista de partidos con información básica

GET /open/games/outstanding

Obtiene el partido destacado (mayor relevancia).

- **Auth**: No requiere
- **Response**: Partido destacado con información completa

OpenClubController (`/open/clubs`)

GET /open/clubs

Lista todos los clubes registrados.

- **Auth**: No requiere
- **Response**: Lista de clubes

GET /open/clubs/{id}

Obtiene información detallada de un club específico.

- **Auth**: No requiere
- **Response**: Club con equipos y jugadores

OpenCompetitionController (`/open/competitions`)

GET /open/competitions

Lista todas las competiciones disponibles.

- **Auth**: No requiere
- **Response**: Lista de competiciones con ligas asociadas

GET /open/competitions/{id}

Obtiene información detallada de una competición.

- **Auth**: No requiere
- **Response**: Competición con ligas y partidos

OpenLeagueController (`/open/leagues`)

GET /open/leagues

Lista todas las ligas disponibles.

- **Auth**: No requiere
- **Response**: Lista de ligas

GET /open/leagues/{id}

Obtiene información detallada de una liga específica.

- **Auth**: No requiere
- **Response**: Liga con clasificación y partidos

WebSocket: Comunicación en Tiempo Real

Configuración WebSocket

Se utiliza STOMP (Simple Text Oriented Messaging Protocol) sobre WebSocket para comunicación bidireccional.

WebSocketConfig

```
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    @Override
    public void configureMessageBroker(MessageBrokerRegistry config) {
        config.enableSimpleBroker("/topic");  // Broadcast
        config.setApplicationDestinationPrefixes("/app");
    }

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/ws")
                .setAllowedOriginPatterns("*")
                .withSockJS();
    }
}
```

RefereeWebSocketController

Endpoint: /app/games/{uniqueCode}/subscribe

Cliente se suscribe a actualizaciones de un partido específico.

- **Topic**: `/topic/games/{uniqueCode}`
- **Eventos emitidos**:
  - `STARTED` - Partido iniciado
  - `POINT_UPDATE` - Puntos actualizados
  - `SET_FINISHED` - Set finalizado
  - `SANCTION_ADDED` - Sanción registrada
  - `FINISHED` - Partido finalizado

LiveGameBroadcastService

Servicio que gestiona el broadcasting de eventos:

```
@Service
public class LiveGameBroadcastService {

    public void broadcastGameUpdate(String uniqueCode, Game game, String
eventType) {
        GameUpdateMessage message = new GameUpdateMessage();
        message.setEventType(eventType);
        message.setGame(game);
        message.setTimestamp(new Date());

        simpMessagingTemplate.convertAndSend(
            "/topic/games/" + uniqueCode,
            message
        );
    }
}
```

Flujo de Comunicación

1. Cliente se conecta: `new SockJS('/ws')`
2. Cliente se suscribe: `/topic/games/{uniqueCode}`
3. Árbitro actualiza puntos: POST `/referee/matches/{uniqueCode}/sets/{setId}/points`
4. Backend procesa y guarda en BD
5. Backend broadcast: `/topic/games/{uniqueCode}` con tipo "POINT_UPDATE"
6. Todos los clientes suscritos reciben actualización instantánea

Servicios Adicionales

ImageController (`/images`)

GET /images/{profilepic}

Obtiene la imagen de perfil del usuario autenticado.

- **Auth**: Requiere JWT
- **Response**: Imagen en bytes (image/jpeg)

POST /images/{profilepic}

Sube una nueva imagen de perfil.

- **Auth**: Requiere JWT
- **Body**: MultipartFile (imagen)
- **Response**: URL de la imagen guardada

ContactController (`/mail/contact`)

POST /mail/contact

Envía un email de contacto desde el formulario público.

- **Auth**: No requiere
- **Body**: `{name, email, message}`
- **Response**: Confirmación de envío

EmailService

Servicio interno para envío de correos electrónicos:

- Verificación de cuenta (con token)
- Recuperación de contraseña (con token)
- Notificaciones de sistema
- Formularios de contacto

HealthController (`/health`)

GET /health

Endpoint de health check para monitoreo.

- **Auth**: No requiere
- **Response**: `OK{1}`

Conclusión

El backend de DOT & DOT implementa una arquitectura REST robusta con comunicación en tiempo real mediante WebSockets, autenticación segura mediante JWT, y un modelo de datos completo que digitaliza el sistema de arbitraje de voleibol.

Tecnologías Utilizadas

- **Spring Boot 3.4.2** - Framework principal
- **Spring Security** - Autenticación y autorización
- **Spring Data JPA** - Persistencia con Hibernate
- **MySQL** - Base de datos relacional
- **JWT** - Tokens de autenticación
- **WebSocket + STOMP** - Comunicación en tiempo real
- **JavaMail** - Envío de correos
- **Lombok** - Reducción de boilerplate

Características Implementadas

1. Sistema completo de autenticación con verificación por email
2. Control de acceso basado en roles (Admin, Referee, User)
3. CRUD completo de todas las entidades del sistema
4. API de arbitraje con validación automática de reglas de voleibol
5. Comunicación en tiempo real mediante WebSockets
6. Endpoints públicos para consulta sin autenticación
7. Gestión de imágenes de perfil
8. Servicio de correo electrónico integrado
9. Broadcasting automático de eventos de partidos
10. Arquitectura en capas (Controller-Service-Repository)
11. Validaciones a nivel de entidad y controlador
12. Manejo centralizado de errores
13. Logging de actividades

Escalabilidad y Mantenimiento

La arquitectura implementada permite:

- Separación de responsabilidades por módulos
- Fácil extensión con nuevos endpoints
- Testing independiente por capas
- Migración a microservicios si es necesario
- Integración con servicios externos