Informe Técnico - API de Apuestas Deportivas Ficticias

Proyecto: Sistema de Gestión de Apuestas Ficticias
Tecnologías: NestJS, TypeScript, PostgreSQL, TypeORM, JWT
Autores: Alejandro Mejía, Nicolás Cardona, Juan Eraso, Carlos Sánchez
Fecha: Octubre 2025

1. Introducción

1.1 Descripción del Proyecto

Este proyecto implementa una API RESTful para un sistema de apuestas deportivas ficticias donde los usuarios pueden apostar dinero virtual en eventos deportivos. El sistema permite gestionar usuarios, eventos y apuestas, implementando mecanismos de autenticación y autorización para proteger los recursos según roles de usuario.

1.2 Objetivos

- Proporcionar endpoints seguros para la gestión de usuarios, eventos y apuestas
- Implementar autenticación mediante tokens JWT
- Establecer un sistema de autorización basado en roles (usuario regular y administrador)
- Garantizar la persistencia confiable de datos mediante PostgreSQL y TypeORM
- Automatizar el cálculo de ganancias al cerrar eventos

2. Arquitectura del Sistema

2.1 Stack Tecnológico

El sistema fue desarrollado utilizando las siguientes tecnologías:

- **Backend Framework:** NestJS 11.x - Framework progresivo de Node.js basado en TypeScript
- **Lenguaje:** TypeScript 5.x - Superset tipado de JavaScript
- **Base de Datos:** PostgreSQL 14.3 - Sistema de gestión de base de datos relacional
- **ORM:** TypeORM 0.3.x - Mapeo objeto-relacional para TypeScript
- **Autenticación:** JWT - JSON Web Tokens para autenticación stateless
- **Validación:** class-validator y class-transformer para validación de DTOs
- **Documentación:** Swagger/OpenAPI para documentación interactiva
- **Testing:** Jest con cobertura superior al 80%
- **Containerización:** Docker para la base de datos

2.2 Arquitectura Modular

La aplicación sigue una arquitectura modular donde cada funcionalidad está encapsulada en módulos independientes:

- **Módulo Auth:** Maneja el registro, inicio de sesión y generación de tokens
- **Módulo Users:** Gestiona la información y balance de usuarios
- **Módulo Events:** Administra eventos deportivos y sus opciones de apuesta
- **Módulo Bets:** Controla la creación y procesamiento de apuestas
- **Módulo Seed:** Proporciona datos de prueba para desarrollo

2.3 Modelo de Datos

El sistema utiliza cuatro entidades principales:

User (Usuario): Almacena información del apostador, incluyendo nombre de usuario, contraseña hasheada, balance de dinero ficticio, roles asignados y estado activo/inactivo. Cada usuario inicia con 10,000 créditos.

Event (Evento): Representa un evento deportivo sobre el cual se puede apostar. Contiene nombre, descripción, estado (abierto/cerrado) y resultado final. Cada evento tiene múltiples opciones de apuesta.

EventOption (Opción de Evento): Define las posibles opciones de resultado para un evento con su respectiva cuota decimal. Por ejemplo, en un partido de fútbol podría haber opciones como "Equipo A gana" con cuota 2.5.

Bet (Apuesta): Registra una apuesta realizada por un usuario. Incluye la opción seleccionada, las odds al momento de apostar, el monto apostado, estado (pendiente/ganada/perdida) y la ganancia calculada.

Las relaciones entre entidades son:

- Un usuario puede tener múltiples apuestas (1:N)
- Un evento puede tener múltiples opciones (1:N)
- Un evento puede tener múltiples apuestas (1:N)

3. Funcionalidades Implementadas

3.1 Gestión de Autenticación

El módulo de autenticación permite a los usuarios registrarse e iniciar sesión de forma segura. Durante el registro, se hashea la contraseña utilizando bcrypt con 10 salt rounds antes de almacenarla. Al iniciar sesión, el sistema verifica las credenciales y genera un token JWT válido por 24 horas que contiene el ID, nombre de usuario y roles del usuario.

3.2 Gestión de Usuarios

Los usuarios pueden ser creados manualmente o mediante el registro. Los administradores tienen permisos para listar todos los usuarios del sistema, mientras que los usuarios regulares solo pueden consultar su propia información. El sistema permite actualizar datos de usuario, consultar el balance actual y eliminar cuentas (solo administradores).

3.3 Gestión de Eventos

Los administradores pueden crear eventos deportivos especificando nombre, descripción y múltiples opciones con sus respectivas cuotas. Los eventos creados están inicialmente en estado "abierto", permitiendo que los usuarios realicen apuestas. Los administradores pueden cerrar eventos estableciendo un resultado ganador, lo cual activa el procesamiento de apuestas. El sistema también permite listar eventos públicamente (solo los abiertos) o ver todos los eventos si se está autenticado.

3.4 Gestión de Apuestas

Los usuarios autenticados pueden crear apuestas sobre eventos abiertos. El sistema valida que el usuario tenga saldo suficiente antes de aceptar la apuesta y descuenta automáticamente el monto del balance. Las apuestas quedan en estado "pendiente" hasta que el evento se cierra. Se registran las odds del momento exacto de la apuesta para garantizar que no cambien posteriormente. Los usuarios pueden consultar su historial de apuestas y estadísticas como total apostado, ganancias y pérdidas.

3.5 Procesamiento de Resultados

Una vez cerrado un evento, los administradores pueden ejecutar el procesamiento de apuestas. El sistema compara cada apuesta pendiente con el resultado ganador: si coinciden, marca la apuesta como "ganada" y calcula la ganancia multiplicando el monto por las odds; si no coincide, marca la apuesta como "perdida". El balance de los usuarios ganadores se actualiza automáticamente sumando el monto apostado más la ganancia.

4. Descripción de Endpoints

4.1 Autenticación

POST /api/auth/register

Permite el registro de nuevos usuarios en el sistema. Recibe nombre de usuario (mínimo 3 caracteres) y contraseña (mínimo 6 caracteres). Opcionalmente se pueden especificar roles, aunque por defecto se asigna el rol "user". Retorna la información del usuario creado junto con un token JWT. La contraseña nunca se retorna en la respuesta.

POST /api/auth/login

Autentica usuarios existentes mediante nombre de usuario y contraseña. Verifica que las credenciales sean correctas y que el usuario esté activo. Si la validación es exitosa, genera y retorna un token JWT que el cliente debe incluir en requests subsecuentes.

4.2 Usuarios

POST /api/users

Crea un usuario manualmente sin requerir autenticación. Permite especificar nombre de usuario, contraseña, roles y balance inicial. Este endpoint es útil para pruebas o creación de usuarios administrativos.

GET /api/users

Lista todos los usuarios registrados. Requiere autenticación y rol de administrador. Retorna un array con la información de cada usuario, excluyendo las contraseñas.

GET /api/users/:id

Obtiene la información completa de un usuario específico mediante su ID UUID. Requiere autenticación pero cualquier usuario autenticado puede consultarlo. Útil para verificar balance y roles.

GET /api/users/:id/balance

Endpoint específico para consultar únicamente el balance actual de un usuario. Retorna un objeto simple con la propiedad balance.

PATCH /api/users/:id

Actualiza parcialmente la información de un usuario. Todos los campos son opcionales: username, password, roles, balance e isActive. Solo se actualizan los campos proporcionados en la petición.

DELETE /api/users/:id

Elimina permanentemente un usuario del sistema. Requiere rol de administrador. Retorna código 204 sin contenido si la operación es exitosa.

4.3 Eventos

POST /api/events

Crea un nuevo evento deportivo. Requiere rol de administrador. Se debe proporcionar nombre del evento (mínimo 5 caracteres), descripción opcional y un array de opciones.

Cada opción debe incluir nombre y cuota decimal (odds). El evento se crea automáticamente en estado "abierto".

GET /api/events

Lista todos los eventos del sistema, tanto abiertos como cerrados. Requiere autenticación. Incluye todas las opciones de cada evento con sus respectivas cuotas.

GET /api/events/open

Lista únicamente los eventos en estado "abierto" que están disponibles para apostar. Este endpoint es público y no requiere autenticación, permitiendo que visitantes vean las oportunidades de apuesta.

GET /api/events/:id

Obtiene los detalles completos de un evento específico incluyendo todas sus opciones. Público, no requiere autenticación.

PATCH /api/events/:id

Actualiza la información de un evento existente. Solo administradores. Permite modificar nombre, descripción y estado.

POST /api/events/:id/close

Cierra un evento estableciendo el resultado ganador. Requiere especificar el resultado final, que debe coincidir exactamente con el nombre de una de las opciones del evento. Una vez cerrado, el evento no acepta más apuestas.

DELETE /api/events/:id

Elimina un evento del sistema. Solo administradores. Falla si existen apuestas asociadas al evento debido a restricciones de integridad referencial.

4.4 Apuestas

POST /api/bets

Crea una nueva apuesta. Requiere autenticación. Parámetros necesarios: ID del usuario apostador, ID del evento, opción seleccionada y monto a apostar. El sistema valida automáticamente que el evento esté abierto, que la opción exista en el evento y que el usuario tenga saldo suficiente. Al crear la apuesta, se descuenta inmediatamente el monto del balance del usuario y se registran las odds actuales.

GET /api/bets

Lista todas las apuestas del sistema con información del usuario y evento asociados. Solo administradores. Útil para auditoría y monitoreo.

GET /api/bets/:id

Obtiene detalles completos de una apuesta específica, incluyendo información del usuario apostador y del evento. Requiere autenticación.

GET /api/bets/user/:userId

Lista todas las apuestas realizadas por un usuario específico. Incluye el estado de cada apuesta y la ganancia obtenida. Requiere autenticación.

GET /api/bets/user/:userId/stats

Retorna estadísticas agregadas de las apuestas de un usuario: total de apuestas, apuestas ganadas, perdidas y pendientes, monto total apostado, ganancias totales y pérdidas totales.

GET /api/bets/event/:eventId

Lista todas las apuestas realizadas sobre un evento específico. Incluye información de los usuarios apostadores. Requiere autenticación.

POST /api/bets/event/:eventId/process

Procesa todas las apuestas pendientes de un evento cerrado. Solo administradores. Compara cada apuesta con el resultado final, marca las ganadoras y perdedoras, calcula ganancias y actualiza balances. Retorna estadísticas del procesamiento: apuestas procesadas, ganadas, perdidas y monto total pagado.

DELETE /api/bets/:id

Elimina una apuesta del sistema. Requiere autenticación.

4.5 Seed

POST /api/seed

Ejecuta un script que puebla la base de datos con datos de prueba: 4 usuarios (1 administrador con credenciales admin/password123 y 3 usuarios regulares), 6 eventos deportivos variados con múltiples opciones, y 8 apuestas de ejemplo. Útil para iniciar rápidamente el desarrollo o testing.

DELETE /api/seed

Limpia completamente la base de datos eliminando todos los registros mediante TRUNCATE CASCADE. Endpoint peligroso que debe usarse solo en desarrollo.

5. Implementación de Autenticación

5.1 Estrategia JWT

El sistema implementa autenticación mediante JSON Web Tokens, lo que permite una arquitectura stateless donde el servidor no necesita mantener sesiones. Esta elección facilita la escalabilidad horizontal y la separación entre frontend y backend.

5.2 Flujo de Autenticación

El proceso comienza cuando un usuario se registra o inicia sesión enviando sus credenciales a los endpoints correspondientes. Durante el registro, la contraseña se procesa con bcrypt utilizando 10 rondas de salt, generando un hash seguro que se almacena en la base de datos. En el inicio de sesión, se recupera el usuario de la base incluyendo el campo password (normalmente excluido) y se compara el hash almacenado con la contraseña proporcionada usando bcrypt.compareSync().

Si las credenciales son válidas, el sistema genera un token JWT firmado con una clave secreta almacenada en variables de entorno. El payload del token contiene únicamente datos no sensibles: ID del usuario, nombre de usuario y roles. El token tiene una expiración de 24 horas.

5.3 Validación de Tokens

Para proteger endpoints, se utiliza el AuthGuard de Passport que intercepta las peticiones y extrae el token del header Authorization (formato: "Bearer token"). El JwtStrategy valida la firma del token, verifica que no haya expirado y extrae el payload. Con el ID del payload, se consulta la base de datos para obtener el usuario completo y se verifica que esté activo. El usuario se adjunta al objeto request, permitiendo que los controladores accedan a él mediante decoradores personalizados.

5.4 Seguridad de Contraseñas

Las contraseñas nunca se almacenan en texto plano. Se utiliza bcrypt con 10 salt rounds, lo que genera hashes únicos incluso para contraseñas idénticas. El campo password en la entidad User tiene la propiedad `select: false` por defecto, evitando que se incluya accidentalmente en respuestas JSON. Solo se selecciona explícitamente cuando es necesario validar credenciales.

5.5 Componentes Técnicos

AuthService: Contiene la lógica de negocio para registro y login. Maneja el hashing de contraseñas, validación de credenciales y generación de tokens.

JwtStrategy: Estrategia de Passport que extiende PassportStrategy(Strategy). Configura la extracción de tokens desde el header Authorization y define la lógica de validación del payload.

JwtModule: Se configura con la clave secreta y opciones de expiración. La clave se obtiene desde variables de entorno para mantener la seguridad.

6. Implementación de Autorización

6.1 Sistema de Roles

El sistema implementa autorización basada en roles (RBAC - Role-Based Access Control) con dos roles definidos: "user" (usuario regular) y "admin" (administrador). Los roles se almacenan como un array en la entidad User, permitiendo que un usuario tenga múltiples roles simultáneamente.

6.2 Flujo de Autorización

Una vez que el usuario está autenticado y el AuthGuard ha validado su token, entra en acción el RolesGuard. Este guard utiliza Reflection para leer metadata asociada a los endpoints mediante decoradores. Si un endpoint tiene roles requeridos especificados, el guard compara los roles del usuario autenticado con los roles permitidos. Si hay coincidencia en al menos un rol, se permite el acceso; de lo contrario, se lanza una excepción ForbiddenException (403).

6.3 Decoradores Personalizados

@Auth(...roles): Decorador compuesto que aplica tanto autenticación como autorización. Acepta roles como parámetros y automáticamente aplica el AuthGuard y RolesGuard. Si no se especifican roles, solo valida autenticación.

```
// Solo usuarios autenticados
@Auth()

// Solo administradores
@Auth(ValidRoles.ADMIN)
```

@RoleProtected(...roles): Establece metadata de roles requeridos usando SetMetadata. Es utilizado internamente por @Auth().

@GetUser(property?): Extrae el usuario del request. Si se proporciona una propiedad, retorna solo ese campo (ej: @GetUser('id') retorna solo el ID).

6.4 Guards Implementados

AuthGuard('jwt'): Guard proporcionado por Passport que valida el token JWT y carga el usuario.

RolesGuard: Guard personalizado que verifica permisos basados en roles. Usa Reflector para obtener los roles requeridos del endpoint. Si no hay roles especificados, permite acceso a cualquier usuario autenticado.

6.5 Protección de Endpoints

Los endpoints se protegen aplicando el decorador @Auth() en los métodos del controlador:

- Endpoints públicos: Sin decorador @Auth()
- Endpoints autenticados: @Auth() sin parámetros
- Endpoints administrativos: @Auth(ValidRoles.ADMIN)

Esta arquitectura permite un control granular de acceso manteniendo el código limpio y declarativo.

7. Persistencia de Datos

7.1 Base de Datos PostgreSQL

Se eligió PostgreSQL como sistema de gestión de base de datos por su robustez, soporte completo de transacciones ACID, integridad referencial y características avanzadas. La base de datos se ejecuta en un contenedor Docker para facilitar el desarrollo y garantizar consistencia entre entornos.

7.2 TypeORM como ORM

TypeORM proporciona el mapeo objeto-relacional, permitiendo definir entidades como clases TypeScript con decoradores. Las ventajas incluyen:

- **Type Safety:** Consultas tipadas que previenen errores en tiempo de desarrollo
- **Migrations:** Control de versiones del esquema de base de datos
- **Relations:** Manejo automático de relaciones entre entidades
- **Query Builder:** Constructor de consultas SQL de manera programática
- **Repositories:** Patrón repository para acceso a datos

7.3 Definición de Entidades

Las entidades se definen mediante decoradores de TypeORM:

@Entity('nombre_tabla'): Define la clase como entidad de base de datos

@PrimaryGeneratedColumn('uuid'): Genera automáticamente IDs únicos UUID

@Column(): Define columnas con opciones de tipo, longitud, default, nullable, etc.

@CreateDateColumn() / @UpdateDateColumn(): Manejo automático de timestamps

@OneToMany() / @ManyToOne(): Define relaciones entre entidades

@BeforeInsert() / @BeforeUpdate(): Hooks que se ejecutan antes de operaciones

7.4 Operaciones de Base de Datos

Repositories: Cada servicio inyecta el repository correspondiente usando @InjectRepository(Entity). Los repositories proporcionan métodos como find(), findOne(), save(), remove(), etc.

Eager Loading: La entidad Event carga automáticamente sus options debido a la configuración `eager: true`. Esto evita consultas adicionales cuando se requieren las opciones.

Cascade Operations: Al guardar un Event, las EventOptions relacionadas se guardan automáticamente gracias a `cascade: true`.

Transacciones: Para operaciones críticas como crear apuestas, se utilizan transacciones explícitas mediante QueryRunner, garantizando atomicidad en el descuento de balance y creación de apuesta.

7.5 Integridad Referencial

Las relaciones tienen configuraciones de integridad:

onDelete: 'CASCADE': Si se elimina un evento, sus opciones se eliminan automáticamente

Foreign Keys: TypeORM genera automáticamente restricciones de clave foránea, evitando referencias huérfanas

7.6 Configuración de Conexión

La conexión a PostgreSQL se configura en el módulo principal usando variables de entorno para host, puerto, usuario, contraseña y nombre de base de datos. La opción `synchronize: true` se usa en desarrollo para sincronizar automáticamente el esquema con las entidades, pero debe deshabilitarse en producción a favor de migrations.

7.7 Seed de Datos

El módulo seed permite poblar rápidamente la base de datos con datos de prueba. Utiliza los repositories para crear usuarios, eventos y apuestas de forma programática. La limpieza se realiza mediante comandos TRUNCATE CASCADE que eliminan todos los registros y reinician secuencias, manteniendo la estructura de tablas intacta.

8. Conclusiones

Se implementó exitosamente un sistema completo de apuestas deportivas ficticias con todas las funcionalidades planteadas. El sistema permite gestionar usuarios con autenticación segura, crear y administrar eventos deportivos con múltiples opciones de apuesta, realizar apuestas con validación de saldo y cálculo automático de ganancias al cerrar eventos.

La implementación de JWT proporciona un mecanismo de autenticación stateless robusto y escalable. Los decoradores personalizados simplifican la aplicación de políticas de seguridad en los controladores.

TypeORM facilita el trabajo con PostgreSQL mediante mapeo objeto-relacional, proporcionando type safety y simplificando operaciones CRUD. Las transacciones garantizan consistencia en operaciones críticas como la creación de apuestas. Las relaciones entre entidades mantienen integridad referencial mediante foreign keys y cascade operations.

El proyecto alcanza más del 80% de cobertura de código mediante tests unitarios y de integración. Los tests unitarios validan la lógica de negocio de forma aislada, mientras que los tests e2e verifican el comportamiento completo de los endpoints.

La documentación automática con Swagger proporciona una interfaz interactiva para explorar y probar todos los endpoints de la API, facilitando el desarrollo frontend y la integración con otros sistemas.