Informe Técnico - Sistema de Apuestas Deportivas "Apostólicos"

Proyecto: Sistema de Gestión de Apuestas Deportivas
Tecnologías: NestJS, Next.js, TypeScript, PostgreSQL, TypeORM, JWT
Autores: Alejandro Mejía, Nicolás Cardona, Juan Eraso, Carlos Sánchez
Universidad: Universidad ICESI
Materia: Computación en Internet III
Fecha: Noviembre 18 del 2025

1. Introducción

1.1 Descripción del Proyecto

Este proyecto implementa una aplicación web full-stack para un sistema de apuestas deportivas 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 una API RESTful segura para la gestión de usuarios, eventos y apuestas
- Desarrollar una interfaz web moderna y responsiva con Next.js
- 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
- Implementar gestión de estado eficiente en el frontend
- 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 - 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
- Testing: Jest con cobertura superior al 80%
- Containerización: Docker para la base de datos

Frontend:

- Framework: Next.js 16.x - Framework de React con SSR
- Lenguaje: TypeScript 5.x
- UI Library: React 19.x
- Gestión de Estado: Zustand 5.0.8 - Librería minimalista de estado global
- Estilos: Tailwind CSS 4.x
- Cliente HTTP: Axios 1.13.x
- Testing E2E: Playwright

2.2 Arquitectura Modular del Backend

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 Arquitectura del Frontend

El frontend implementa:

- **Routing:** Sistema de rutas basado en archivos de Next.js App Router
- **Componentes:** Elementos reutilizables organizados por funcionalidad
- **Servicios:** Capa de comunicación con la API backend
- **Estado:** Gestión mediante Zustand stores (auth, events, bets, users, toast)
- **Persistencia:** Middleware de persistencia para mantener la sesión de autenticación

2.4 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.

Relaciones entre entidades:

- 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. Implementación de Autenticación

3.1 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.

3.2 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.

3.3 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.

3.4 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.

3.5 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.

3.6 Autenticación en el Frontend

En el frontend, se implementó una combinación de servicio HTTP y Zustand store:

authService: Maneja las peticiones HTTP de login y registro hacia el backend.

auth.store (Zustand): Gestiona el estado global de autenticación usando el middleware `persist` para almacenar automáticamente el token JWT y datos del usuario en localStorage. El store expone métodos para login, registro, logout y verificación de estado.

Interceptores Axios: Inyectan automáticamente el token en todas las peticiones HTTP mediante el header Authorization. Si se recibe un error 401, se limpia la sesión y se redirige al login automáticamente.

4. Implementación de Autorización

4.1 Sistema de Roles (RBAC)

El sistema implementa Role-Based Access Control (RBAC) con dos roles:

USER: Puede realizar apuestas, consultar sus propias apuestas, ver su historial y balance, y listar eventos activos.

ADMIN: Posee todos los permisos de USER, además de crear/actualizar/eliminar eventos, cerrar eventos, seleccionar ganadores, procesar resultados de apuestas, ver todas las apuestas del sistema y gestionar usuarios.

4.2 Flujo de Autorización

Al registrarse, cada usuario recibe automáticamente el rol USER. Solo administradores pueden asignar el rol ADMIN a otros usuarios.

Cuando un usuario autenticado intenta acceder a un endpoint protegido:

1. El AuthGuard verifica la autenticación
2. El RolesGuard se activa después del AuthGuard
3. Se obtienen los roles requeridos del endpoint mediante metadata (decorador Roles)
4. Se compara el rol del usuario con los roles permitidos
5. Si hay coincidencia, se otorga acceso; de lo contrario, se retorna HTTP 403 Forbidden

4.3 Decoradores Personalizados

@Roles(Role.ADMIN):
Decorador que utiliza SetMetadata para marcar métodos o controladores con los roles requeridos. Permite especificar uno o más roles.

@Auth(Role.ADMIN):
Decorador compuesto que aplica automáticamente AuthGuard y RolesGuard. Simplifica la protección de endpoints al combinar autenticación y autorización en una sola línea.

@GetUser():
Extrae el usuario autenticado del objeto request. Permite acceder directamente al usuario en los métodos del controlador sin necesidad de acceder manualmente a `req.user`.

4.4 Guards

RolesGuard:
Guard personalizado que implementa CanActivate. Utiliza Reflector para leer la metadata de roles del endpoint. Accede al usuario desde request (inyectado por AuthGuard) y valida si alguno de sus roles coincide con los requeridos. Si no hay roles especificados en la metadata, permite el acceso (endpoint público después de autenticación).

4.5 Protección de Endpoints

Públicos: Login y registro no requieren decoradores de autenticación.

Autenticados: Endpoints como crear apuesta usan `@Auth()` sin roles específicos, permitiendo acceso a cualquier usuario autenticado.

Administrativos: Endpoints como crear eventos, cerrar eventos y procesar resultados usan `@Auth(Role.ADMIN)`, restringiendo acceso solo a administradores.

5. Interfaz de Usuario

5.1 Diseño y Estructura

La interfaz se construyó con Tailwind CSS 4.x, priorizando responsividad y usabilidad. Las páginas de autenticación incluyen formularios de login y registro con validación en tiempo real, mostrando errores de forma contextual junto a cada campo.

El dashboard principal integra una barra de navegación con información del usuario, lista de eventos deportivos disponibles, panel deslizante para gestionar apuestas y área de contenido dinámico. El panel de administración proporciona secciones específicas para gestionar eventos y procesar resultados.

5.2 Arquitectura de Componentes

La aplicación sigue el principio de composición, donde elementos complejos se crean combinando componentes más simples. Los componentes se organizan en tres categorías:

Componentes de Bajo Nivel: Botones, inputs, cards que encapsulan estilos básicos.

Componentes de Negocio: EventsList, BetSlip, Navbar con lógica específica del dominio.

Componentes de Layout: Sidebar, Container que organizan la estructura visual.

5.3 Renderizado Condicional por Roles

El frontend utiliza renderizado condicional para mostrar u ocultar elementos según el rol del usuario. Esta capa mejora la experiencia del usuario, pero la seguridad real está garantizada por la validación en el backend mediante guards.

6. Gestión del Estado

6.1 Estrategia Adoptada: Zustand

El proyecto utiliza Zustand 5.0.8 como gestor de estado global. Zustand se seleccionó por su simplicidad, rendimiento y API minimalista que no requiere boilerplate excesivo. A diferencia de Redux, Zustand no necesita providers, reducers ni actions explícitas, lo que reduce significativamente la complejidad del código.

6.2 Arquitectura de Stores

Se implementaron cinco stores especializados:

auth.store: Gestiona autenticación (usuario, token, login, registro, logout). Utiliza el middleware `persist` de Zustand para mantener la sesión en localStorage, permitiendo que persista entre recargas de página.

events.store: Administra eventos deportivos (lista de eventos, crear, actualizar, cerrar, eliminar). Incluye optimistic updates para reflejar cambios inmediatamente en la UI.

bets.store: Controla apuestas del usuario (historial de apuestas, crear nueva apuesta). Se sincroniza con el backend después de cada operación.

users.store: Gestiona información de usuarios (para vistas administrativas).

toast.store: Maneja notificaciones toast para feedback visual (success, error, info, warning) sin necesidad de alertas modales.

6.3 Patrón de Implementación

Cada store define su estado y acciones dentro de una función `create()`. Las acciones son métodos asíncronos que:

1. Establecen `isLoading: true` al iniciar la operación
2. Ejecutan la llamada al servicio HTTP correspondiente
3. Actualizan el estado con los datos recibidos
4. Manejan errores mostrando mensajes mediante el toast store
5. Establecen `isLoading: false` al finalizar

6.4 Sincronización con el Backend

Las operaciones CRUD siguen este flujo:

1. El componente invoca una acción del store (ej: `createEvent()`)
2. El store llama al servicio HTTP correspondiente
3. Si la operación es exitosa, el store actualiza su estado local inmediatamente
4. Se muestra un toast de confirmación
5. En caso de error, se captura y se muestra mediante toast.error()

Para operaciones críticas como crear apuestas, se implementa actualización inmediata del balance del usuario antes de la confirmación del servidor, mejorando la percepción de velocidad.

6.5 Interceptores HTTP

Los interceptores de Axios centralizan la lógica de comunicación:

Request Interceptor: Inyecta automáticamente el token JWT desde el auth.store en el header Authorization de todas las peticiones.

Response Interceptor: Maneja errores 401 (limpia sesión y redirige al login) y errores 403 (muestra toast de error), garantizando una experiencia consistente en el manejo de errores de autenticación/autorización.

7. Funcionalidades Principales

7.1 Gestión de Usuarios

Registro: Al registrarse, el sistema valida que el username sea único, verifica que la contraseña tenga mínimo 6 caracteres, la encripta con bcrypt, asigna automáticamente el rol USER y establece un balance inicial de $10,000 COP.

Perfil: Los usuarios autenticados consultan su información personal (username, email), balance actual, rol asignado y fecha de registro.

7.2 Gestión de Eventos Deportivos

Estructura: Cada evento contiene nombre, descripción, estado (abierto/cerrado/cancelado), múltiples opciones de apuesta con sus cuotas (multiplicador de ganancia) y la opción ganadora una vez procesado.

Creación (Solo Administradores): Los administradores definen el nombre del evento, descripción, opciones de apuesta y cuotas asociadas a cada opción.

Consulta: Vista pública muestra eventos abiertos disponibles para apostar. Vista administrativa muestra todos los eventos independientemente del estado.

Cierre (Solo Administradores): El administrador selecciona la opción ganadora, el sistema cambia el estado a "cerrado" y procesa automáticamente el resultado de todas las apuestas asociadas.

7.3 Sistema de Apuestas

Creación: El proceso valida que el usuario tenga balance suficiente, que el evento esté abierto, que la opción pertenezca al evento y que el monto sea mínimo $1,000 COP. Al crear una apuesta, se descuenta el monto del balance, se registra con estado "pendiente", se almacena la referencia al usuario/evento/opción y se calcula la ganancia potencial (monto × cuota).

Consultas: Vista personal (usuario consulta solo sus apuestas), vista filtrada por estado (pendientes/ganadas/perdidas), vista administrativa (todas las apuestas) y vista por evento.

Procesamiento de Resultados: Al cerrar un evento, se identifica la opción ganadora y se procesan todas las apuestas: si apostó a la ganadora, estado → "ganada" y se acredita (monto × cuota) al balance; si apostó a otra opción, estado → "perdida" sin acreditación.

7.4 Gestión de Balance

El balance se consulta en tiempo real. Al crear una apuesta, el monto se descuenta atómicamente (si la transacción falla, el balance no se modifica). Al ganar, se acredita automáticamente la ganancia calculada al balance del usuario.
Cuando se procesa una apuesta ganadora, se acredita el monto calculado como: `monto_apostado × cuota_opcion`.

Integridad Transaccional:
Todas las operaciones sobre el balance se ejecutan dentro de transacciones de base de datos, garantizando consistencia incluso ante fallos.

7.5 Sistema de Datos de Prueba (Seed)

Para facilitar el desarrollo y testing, se implementó un sistema de población de datos:

Datos Generados:

- 4 usuarios de prueba (1 administrador, 3 usuarios regulares)
- 6 eventos deportivos de diferentes categorías
- 8 apuestas de ejemplo con variados estados

Utilidad:
Permite inicializar rápidamente una instancia del sistema con datos representativos para realizar pruebas manuales, demostraciones o desarrollo de nuevas funcionalidades.

Limpieza:
También se proporciona funcionalidad para limpiar todos los datos, restableciendo la base de datos a su estado inicial.

8. Pruebas

8.1 Estrategia de Testing

Se implementó una estrategia multicapa: pruebas unitarias (funciones individuales aisladas), pruebas de integración (interacción entre módulos) y pruebas End-to-End (simulación de comportamiento real del usuario).

8.2 Herramientas

Jest: Framework de testing para el backend. Proporciona ejecución paralela, sistema de mocking, reportes de cobertura y sintaxis expresiva.

Supertest: Librería para testing de APIs HTTP, permite simular peticiones sin levantar servidor completo.

Playwright: Testing E2E que automatiza navegadores reales (Chrome, Firefox, Safari), simula interacciones de usuario y genera reportes detallados.

8.3 Pruebas del Backend

Servicios: Validación de lógica de negocio (métodos de autenticación, operaciones CRUD, cálculos de ganancias).

Controladores: Verificación de códigos HTTP, formato de respuesta y manejo de errores.

Guards y Estrategias: Validación de tokens JWT, aplicación de roles y rechazo de accesos no autorizados.

8.4 Pruebas E2E

Flujo de Autenticación: Registro → Redirección a login → Inicio de sesión → Acceso al dashboard.

Flujo de Apuestas: Visualización de eventos → Selección de opción → Ingreso de monto → Confirmación → Validación de saldo → Descuento del balance → Apuesta en historial.

Flujo Administrativo: Login como admin → Gestión de eventos → Creación de evento → Cierre de evento → Procesamiento automático de apuestas.

8.5 Cobertura

Se alcanzó más del 80% de cobertura en el backend, cubriendo statements, branches, functions y lines.

9. Instalación

9.1 Requisitos

- Node.js 18+ o Bun
- PostgreSQL 13+ o Docker
- Git

9.2 Backend

Instalar dependencias, crear archivo `.env` con credenciales de PostgreSQL, clave secreta JWT y puerto, iniciar PostgreSQL mediante Docker Compose y ejecutar el servidor en modo desarrollo poblando la base con datos de prueba.

9.3 Frontend

Instalar dependencias, configurar URL del backend en `.env` e iniciar el servidor de desarrollo Next.js.

9.4 Acceso

Frontend: http://localhost:3001
API Backend: http://localhost:3000
Documentación API: http://localhost:3000/api

10. Conclusiones

El proyecto implementó exitosamente un sistema de apuestas deportivas con autenticación basada en JWT (tokens firmados con expiración de 24 horas, contraseñas encriptadas con bcrypt) y autorización mediante RBAC con dos roles (USER y ADMIN). El control de acceso se implementó con guards personalizados que validan permisos antes de ejecutar operaciones.

La interfaz de usuario es responsiva, construida con Next.js y Tailwind CSS, con validaciones en tiempo real y feedback visual contextual mediante toasts. La gestión del estado utiliza Zustand 5.0.8, una librería minimalista que simplifica el manejo de estado global sin el boilerplate de Redux. Se implementaron cinco stores especializados (auth, events, bets, users, toast) con middleware de persistencia para la sesión de usuario.

El sistema cubre funcionalidades completas de gestión de usuarios, eventos deportivos, apuestas y procesamiento de resultados. Se alcanzó más del 80% de cobertura en pruebas del backend (Jest, Supertest) y se implementaron pruebas E2E con Playwright validando flujos críticos de autenticación, apuestas y administración.

La arquitectura modular del backend (NestJS con TypeORM) y la separación frontend-backend con gestión de estado clara mediante Zustand facilitan el mantenimiento y escalabilidad del sistema.

11. Referencias

- NestJS Documentation: https://docs.nestjs.com/
- Next.js Documentation: https://nextjs.org/docs
- Zustand Documentation: https://zustand-demo.pmnd.rs/
- TypeORM: https://typeorm.io/
- Jest: https://jestjs.io/
- Playwright: https://playwright.dev/

Repositorio del Proyecto:
https://github.com/Carlos-SD/TNextJSCompunetIII

Universidad ICESI
Computación en Internet III
Noviembre del 2025