INFORME TÉCNICO - GYM

Taller: Backend – NestJS

Docente: Gustavo González

Integrantes:

- **Luis Manuel Rojas Correa**
- **Santiago Angel Ordoñez**
- **Cristian Camilo Molina**
- **Juan Camilo Corrales Osvath**

1. INTRODUCCIÓN

El presente informe documenta el desarrollo del Gym, una aplicación backend robusta construida con NestJS, TypeORM y PostgreSQL. El sistema implementa un conjunto completo de funcionalidades para la gestión de un gimnasio, incluyendo autenticación basada en JWT, autorización por roles, gestión de membresías, suscripciones y control de asistencias.La API fue diseñada siguiendo los principios de arquitectura REST, implementando buenas prácticas de desarrollo como la separación de responsabilidades, validación de datos, manejo de errores y documentación con Swagger.

2. ARQUITECTURA DEL SISTEMA

2.1 Stack Tecnológico

- Framework: NestJS
- Lenguaje: TypeScript
- Base de datos: PostgreSQL 15
- ORM: TypeORM
- Autenticación: JWT (JSON Web Tokens)
- Validación: class-validator y class-transformer
- Documentación: Swagger/OpenAPI
- Containerización: Docker y Docker Compose

2.2 Estructura de Módulos

La aplicación está organizada en los siguientes módulos principales:

1. Auth Module: Gestión de autenticación y usuarios
2. Memberships Module: Administración de planes de membresía

3. Subscriptions Module: Gestión de suscripciones de usuarios
4. Attendances Module: Control de asistencias al gimnasio
5. Seed Module: Población inicial de la base de datos

---

3. IMPLEMENTACIÓN DE AUTENTICACIÓN

3.1 Estrategia JWT

El sistema de autenticación se basa en JSON Web Tokens (JWT), implementado mediante Passport y la estrategia JWT. La configuración permite tokens firmados con el algoritmo HS256 y una clave secreta almacenada en variables de entorno.Flujo de autenticación:

1. El usuario envía sus credenciales (email y contraseña) al endpoint de login
2. El sistema valida las credenciales contra la base de datos
3. Las contraseñas están hasheadas usando bcrypt para seguridad
4. Si las credenciales son válidas, se genera un token JWT que incluye el ID y email del usuario
5. El cliente debe enviar este token en el header Authorization de las peticiones subsecuentes
6. La estrategia JWT valida el token en cada petición protegida y adjunta el usuario al request

Implementación técnica: La clase JwtStrategy extiende PassportStrategy y valida el payload del token. En el método validate, se verifica que el usuario exista en la base de datos, se cargan sus roles mediante una relación eager, y se valida que el usuario esté activo. Si alguna validación falla, se lanza una UnauthorizedException.

3.2 Endpoints de Autenticación

POST /auth/register

Registra un nuevo usuario en el sistema.

Parámetros:

- email (string, requerido): Correo electrónico único del usuario
- fullName (string, requerido): Nombre completo
- age (number, requerido): Edad entre 1 y 120 años
- password (string, requerido): Contraseña de 6-50 caracteres

Respuestas:

- 201: Usuario registrado exitosamente, retorna el usuario creado con su token JWT
- 400: Email ya está en uso o datos inválidos

POST /auth/login

Autentica un usuario existente.

Parámetros:

- email (string, requerido)
- password (string, requerido)

Respuestas:

- 200: Login exitoso, retorna token JWT y datos del usuario
- 401: Credenciales inválidas

GET /auth

Obtiene la lista de todos los usuarios del sistema.

Autorización: Requiere rol de administrador

Respuestas:

- 200: Lista de usuarios
- 403: Usuario no autorizado

GET /auth/:id

Obtiene un usuario específico por su ID.

Parámetros:

- id (UUID, requerido): Identificador único del usuario

Autorización: Requiere rol de administrador

Respuestas:

- 200: Datos del usuario
- 404: Usuario no encontrado
- 403: No autorizado

PATCH /auth/:id

Actualiza los datos de un usuario.

Parámetros:

- id (UUID, requerido)
- Campos opcionales: email, fullName, age, password

Autorización: El usuario puede actualizar su propio perfil o requiere ser administrador

Respuestas:

- 200: Usuario actualizado exitosamente

- 403: No tiene permisos para actualizar este usuario
- 404: Usuario no encontrado

DELETE /auth/:id

Elimina un usuario del sistema.

Parámetros:

- id (UUID, requerido)

Autorización: Solo administradores

Respuestas:

- 204: Usuario eliminado exitosamente
- 403: No autorizado
- 404: Usuario no encontrado

4. IMPLEMENTACIÓN DE AUTORIZACIÓN

4.1 Sistema de Roles

El sistema implementa cuatro roles diferentes con distintos niveles de acceso:

1. admin: Acceso total al sistema
2. receptionist: Gestión de membresías, suscripciones y asistencias
3. coach: Consulta de información y estadísticas
4. client: Acceso limitado a su propia información

4.2 Mecanismo de Autorización

La autorización se implementa mediante un decorador personalizado @Auth que combina dos guards:AuthGuard: Verifica que el token JWT sea válido usando Passport.UserRoleGuard: Valida que el usuario autenticado tenga al menos uno de los roles requeridos para acceder al endpoint.El UserRoleGuard utiliza el Reflector de NestJS para obtener los roles permitidos definidos en los metadatos del endpoint. Luego compara estos roles con los roles del usuario autenticado. Si el usuario no tiene ningún rol válido, se lanza una ForbiddenException.

4.3 Estructura de Roles en Base de Datos

Los roles se almacenan en una tabla separada con relación ManyToMany con usuarios. Esto permite que un usuario tenga múltiples roles si es necesario. La tabla intermedia user_roles gestiona esta relación.

5. GESTIÓN DE MEMBRESÍAS

5.1 Descripción

El módulo de membresías administra los diferentes planes o tipos de membresía que el gimnasio ofrece. Estas funcionan como plantillas que pueden ser asignadas a las suscripciones de los usuarios.

5.2 Endpoints

GET /memberships

Lista todas las membresías disponibles ordenadas por fecha de creación.Autorización: admin, receptionist

Respuestas:

- 200: Array de membresías con sus propiedades completas
- 403: No autorizado

GET /memberships/:id

Obtiene los detalles de una membresía específica.

Parámetros:

- id (UUID, requerido)

Autorización: admin, receptionist, coach

Respuestas:

- 200: Datos de la membresía
- 404: Membresía no encontrada
- 403: No autorizado

POST /memberships

Crea una nueva membresía.

Parámetros:

- name (string, requerido): Nombre único de la membresía
- cost (number, requerido): Costo con máximo 2 decimales
- max_classes_assistance (number, requerido): Número máximo de clases
- max_gym_assistance (number, requerido): Número máximo de asistencias al gimnasio
- duration_months (number, requerido): Duración en meses, debe ser 1 o 12
- status (boolean, opcional): Estado activo/inactivo, por defecto true

Autorización: Solo admin

Respuestas:

- 201: Membresía creada exitosamente
- 400: Datos inválidos o nombre duplicado
- 403: No autorizado

PUT /memberships/:id

Actualiza una membresía existente.

Parámetros:

- id (UUID, requerido)
- Campos opcionales: name, cost, max_classes_assistance, max_gym_assistance, duration_months, status

Autorización: Solo admin

Validaciones:

- Si se actualiza el nombre, verifica que no exista otra membresía con ese nombre

Respuestas:

- 200: Membresía actualizada
- 404: Membresía no encontrada
- 400: Nombre duplicado
- 403: No autorizado

PATCH /memberships/:id/toggle-status

Cambia el estado activo/inactivo de una membresía.

Parámetros:

- id (UUID, requerido)

Autorización: Solo admin

Respuestas:

- 200: Estado actualizado
- 404: Membresía no encontrada
- 403: No autorizado

DELETE /memberships/:id

Elimina una membresía mediante soft delete.

Parámetros:

- id (UUID, requerido)

Autorización: Solo admin

Respuestas:

- 204: Membresía eliminada (soft delete)
- 404: Membresía no encontrada
- 403: No autorizado

6. GESTIÓN DE SUSCRIPCIONES

6.1 Descripción

El módulo de suscripciones gestiona las suscripciones activas de los usuarios. Una suscripción vincula a un usuario con una o más membresías y controla las reglas de negocio relacionadas.

6.2 Validaciones de Negocio

1. Un usuario solo puede tener una suscripción activa a la vez
2. Las suscripciones pueden activarse y desactivarse
3. Las suscripciones soportan soft delete

6.3 Endpoints

GET /subscriptions/user/:userId

Obtiene la suscripción de un usuario específico.

Parámetros:

- userId (UUID, requerido)

Autorización: admin, receptionist

Respuestas:

- 200: Datos de la suscripción con relaciones cargadas
- 404: Usuario o suscripción no encontrada
- 403: No autorizado

POST /subscriptions

Crea una nueva suscripción para un usuario.

Parámetros en body:

- userId (UUID, requerido)

Autorización: admin, receptionist

Validaciones:

- Verifica que el usuario exista
- Valida que el usuario no tenga otra suscripción activa

Respuestas:

- 201: Suscripción creada (inicialmente vacía, sin membresías)
- 404: Usuario no encontrado
- 409: Usuario ya tiene una suscripción activa
- 403: No autorizado

POST /subscriptions/:id/memberships

Agrega una membresía a una suscripción existente.

Parámetros de ruta:

- id (UUID, requerido): ID de la suscripción

Parámetros de body:

- membershipId (UUID, requerido)

Autorización: admin, receptionist

Lógica:

- Busca la membresía plantilla
- Agrega la membresía a la suscripción
- Actualiza los valores acumulados: cost, max_classes_assistance, max_gym_assistance
- El duration_months toma el valor máximo entre el actual y el nuevo

Respuestas:

- 200: Membresía agregada exitosamente
- 404: Suscripción o membresía no encontrada
- 403: No autorizado

GET /subscriptions

Lista todas las suscripciones del sistema.

Autorización: Solo admin

Respuestas:

- 200: Array de suscripciones con relaciones (user, memberships)
- 403: No autorizado

GET /subscriptions/:id

Obtiene una suscripción específica por ID.Parámetros:

- id (UUID, requerido)

Autorización: admin, receptionist, clientRespuestas:

- 200: Datos de la suscripción
- 404: Suscripción no encontrada
- 403: No autorizado

PATCH /subscriptions/:id

Actualiza una suscripción.

Parámetros:

- id (UUID, requerido)
- Campos opcionales en body

Autorización: Solo admin

Respuestas:

- 200: Suscripción actualizada
- 404: Suscripción no encontrada
- 403: No autorizado

PATCH /subscriptions/:id/deactivate

Desactiva una suscripción cambiando is_active a false.

Parámetros:

- id (UUID, requerido)

Autorización: Solo admin

Uso: Se utiliza cuando una suscripción expira o se cancela

Respuestas:

- 200: Suscripción desactivada
- 404: Suscripción no encontrada
- 403: No autorizado

PATCH /subscriptions/:id/activate

Activa una suscripción cambiando is_active a true.

Parámetros:

- id (UUID, requerido)

Autorización: Solo admin

Validaciones:

- Verifica que el usuario no tenga otra suscripción activa

Respuestas:

- 200: Suscripción activada
- 409: Usuario ya tiene otra suscripción activa
- 404: Suscripción no encontrada
- 403: No autorizado

DELETE /subscriptions/:id

Elimina una suscripción mediante soft delete.

Parámetros:

- id (UUID, requerido)

Autorización: Solo admin

Respuestas:

- 204: Suscripción eliminada
- 404: Suscripción no encontrada
- 403: No autorizado

7. GESTIÓN DE ASISTENCIAS

7.1 Descripción

El módulo de asistencias controla el check-in y check-out de usuarios al gimnasio, mantiene un historial de asistencias y proporciona estadísticas.

7.2 Endpoints

POST /attendances/check-in

Registra la entrada de un usuario al gimnasio.

Parámetros:

- userId (UUID, requerido)

Autorización: receptionist

Validaciones:

- Usuario debe tener una suscripción activa
- Usuario no debe estar ya en el gimnasio

Respuestas:

- 201: Check-in exitoso
- 400: Usuario ya está dentro o no tiene suscripción activa
- 403: No autorizado

POST /attendances/check-out

Registra la salida de un usuario del gimnasio.

Parámetros:

- userId (UUID, requerido)

Autorización: receptionist

Validaciones:

- Usuario debe estar actualmente dentro del gimnasio

Respuestas:

- 200: Check-out exitoso
- 400: Usuario no está actualmente dentro
- 403: No autorizado

GET /attendances/status/:userId

Consulta el estado actual de asistencia de un usuario.

Parámetros:

- userId (UUID, requerido)

Autorización: admin

Respuestas:

- 200: Estado actual (dentro/fuera del gimnasio)
- 404: Usuario no encontrado
- 403: No autorizado

GET /attendances/history/:userId

Obtiene el historial de asistencias de un usuario.

Parámetros de ruta:

- userId (UUID, requerido)

Query parameters opcionales:

- startDate: Fecha de inicio del rango
- endDate: Fecha fin del rango

Autorización: admin

Respuestas:

- 200: Array de registros de asistencia
- 404: Usuario no encontrado
- 403: No autorizado

GET /attendances/stats/:userId

Calcula estadísticas de asistencia de un usuario.

Parámetros:

- userId (UUID, requerido)

Autorización: admin

Respuestas:

- 200: Estadísticas (total de visitas, promedio de duración, etc.)
- 404: Usuario no encontrado
- 403: No autorizado

GET /attendances/active

Lista todos los usuarios actualmente dentro del gimnasio.

Autorización: admin

Respuestas:

- 200: Array de asistencias activas
- 403: No autorizado

8. MÓDULO SEED

8.1 Descripción

El módulo seed permite poblar la base de datos con datos iniciales para desarrollo y pruebas.

8.2 Endpoint

GET /seed

Ejecuta el proceso de seed completo.

Autorización: No requiere (solo para desarrollo)

Proceso:

1. Limpia todas las tablas (users, roles, memberships)
2. Inserta los 4 roles del sistema
3. Crea usuarios de prueba con sus roles asignados
4. Crea 5 membresías de ejemplo

Datos creados:

Usuarios:

- admin@example.com / admin123 (rol: admin)
- coach@example.com / coach123 (rol: coach)
- client@example.com / client123 (rol: client)
- receptionist@example.com / recep123 (rol: receptionist)

Membresías:

- Básica Mensual: $50, 8 clases, 15 visitas gym, 1 mes
- Premium Mensual: $80, 20 clases, 30 visitas gym, 1 mes
- Básica Anual: $500, 96 clases, 180 visitas gym, 12 meses
- Premium Anual: $800, 240 clases, 365 visitas gym, 12 meses
- VIP Anual: $1200, 365 clases, 365 visitas gym, 12 meses

Respuestas:

- 200: "SEED EXECUTED SUCCESSFULLY"

9. PERSISTENCIA EN BASE DE DATOS

9.1 Configuración de TypeORM

La conexión a la base de datos se configura en el AppModule utilizando TypeORM. Los parámetros de conexión se obtienen de variables de entorno para mayor seguridad y flexibilidad:

- Host: localhost
- Puerto: 5433
- Base de datos: workshop_db
- Usuario: admin
- Contraseña: password123

La configuración incluye:

- autoLoadEntities: true (carga automática de entidades)
- synchronize: true (sincronización automática del schema, solo para desarrollo)

9.2 Entidades y Relaciones

User Entity

Representa a los usuarios del sistema.

Campos:

- id: UUID, clave primaria auto-generada
- email: string, único
- fullName: string
- age: number
- password: string, hasheado con bcrypt
- isActive: boolean, por defecto true
- created_at: timestamp automático
- updated_at: timestamp automático

Relaciones:

- ManyToMany con Role (tabla intermedia: user_roles)
- OneToMany con Subscription

Role Entity

Define los roles disponibles en el sistema.

Campos:

- id: UUID
- name: string (admin, coach, client, receptionist)

Relaciones:

- ManyToMany con User

Membership Entity

Plantillas de membresías del gimnasio.

Campos:

- id: UUID
- name: string, único
- cost: numeric(10,2)
- status: boolean, por defecto true
- max_classes_assistance: integer
- max_gym_assistance: integer
- duration_months: integer (1 o 12)
- created_at: timestamp
- updated_at: timestamp
- deleted_at: timestamp (para soft delete)

Relaciones:

- ManyToMany con Subscription

Subscription Entity

Suscripciones activas de usuarios.

Campos:

- id: UUID
- name: string, único
- cost: numeric(10,2)
- max_classes_assistance: integer
- max_gym_assistance: integer
- duration_months: integer
- purchase_date: date
- is_active: boolean, por defecto true
- created_at: timestamp
- updated_at: timestamp
- deleted_at: timestamp (para soft delete)

Relaciones:

- ManyToOne con User (con onDelete: CASCADE)
- ManyToMany con Membership (tabla intermedia: subscription_memberships)

Attendance Entity

Registros de asistencias al gimnasio.

Campos:

- id: UUID
- userId: UUID (relación con User)
- checkInTime: timestamp
- checkOutTime: timestamp, nullable
- created_at: timestamp
- updated_at: timestamp

9.3 Soft Delete

Las entidades Membership y Subscription implementan soft delete mediante el decorador @DeleteDateColumn. Esto significa que cuando se "elimina" un registro, no se borra físicamente de la base de datos, sino que se marca con una fecha en el campo deleted_at. TypeORM automáticamente excluye estos registros de las consultas normales.

Ventajas del soft delete:

- Permite recuperar datos eliminados accidentalmente
- Mantiene integridad referencial

- Facilita auditorías y trazabilidad

9.4 Validaciones a Nivel de Base de Datos

TypeORM aplica automáticamente las restricciones definidas en las entidades:

- Claves primarias (UUID auto-generadas)
- Campos únicos (email, nombres de membresías y suscripciones)
- Tipos de datos específicos (numeric para montos, date para fechas)
- Constraints de integridad referencial (foreign keys)
- Valores por defecto

10. CONCLUSIONES

La Temple Gym API representa una implementación completa y profesional de un backend moderno utilizando NestJS.

El proyecto cumple con todos los requisitos establecidos:

Autenticación (5%):

- Sistema JWT implementado correctamente
- Login y registro funcionales
- Rutas protegidas con guards

Autorización (5%):

- Cuatro roles distintos definidos
- Permisos por rol correctamente aplicados
- Mecanismo de guards personalizado

Persistencia (10%):

- TypeORM configurado con PostgreSQL
- Relaciones entre entidades bien definidas
- Soft delete implementado

Funcionalidades (25%):

- CRUD completo de membresías
- Gestión de suscripciones con validaciones de negocio
- Control de asistencias
- Sistema de seed

Pruebas (25%):

- Pruebas unitarias implementadas
- Pruebas E2E con cobertura de flujos principales

- Utilidades de testing desarrolladas

Documentación y Despliegue:

- Swagger completamente configurado
- Docker Compose funcional
- Variables de entorno correctamente gestionadas

El código sigue las mejores prácticas de desarrollo, con separación clara de responsabilidades, validación robusta de datos, manejo apropiado de errores y una estructura modular que facilita el mantenimiento y escalabilidad futura del proyecto.