Documentación de la API Backend de Fittrack

Autor: Sergio Javier Lorente Guerra

Base URL (entorno de desarrollo):
https://sergiolg25.iesmontenaranco.com:8000
(prefijo /api en la mayoría de rutas)

Autenticación: JWT Bearer (cabecera Authorization: Bearer <token>)

Índice

4. Gestión de Bloques (Entrenador/Admin)

4.1. POST /api/admin/bloques

4.2. GET /api/admin/bloques

4.3. PUT /api/admin/bloques/{bloqueId}

4.4. DELETE /api/admin/bloques/{bloqueId}

4.5. POST /api/admin/bloques/{bloqueId}/asignar/{usuarioId}

4.6. DELETE /api/admin/bloques/{bloqueId}/desasignar/{usuarioId}

5. Gestión de Semanas (Entrenador)

5.1. POST /api/bloques/{bloqueId}/semanas

5.2. GET /api/bloques/{bloqueId}/semanas

5.3. GET /api/semanas/{semanaId}

5.4. PUT /api/semanas/{semanaId}

5.5. DELETE /api/semanas/{semanaId}

7. Gestión de Ejercicios Sugeridos (Entrenador)

7.1. POST /api/subbloques/{subBloqueId}/ejercicios-sugeridos

7.2. GET /api/subbloques/{subBloqueId}/ejercicios-sugeridos

7.3. GET /api/subbloques/ejercicios-sugeridos/{id}

7.4. PUT /api/subbloques/ejercicios-sugeridos/{id}

7.5. DELETE /api/subbloques/ejercicios-sugeridos/{id}

8. Gestión de Series Sugeridas (Entrenador)

8.1. GET /api/series-sugeridas/ejercicios-sugeridos/{ejercicioId}/series

8.2. POST /api/series-sugeridas/ejercicios-sugeridos/{id}/series

8.3. PUT /api/series-sugeridas/series-sugeridas/{id}

8.4. DELETE /api/series-sugeridas/series-sugeridas/{id}

8.5. GET /api/series-sugeridas/usuarios/{usuarioId}/subbloques/{subBloqueId}/ejercicios/{ejercicioId}/series

9. Endpoints de Usuario (consumo de bloques/semana/sub- bloque/ejercicios)

9.1. GET /api/usuarios/{usuarioId}/bloques

9.2. GET /api/usuarios/{usuarioId}/bloques/{bloqueId}/semanas

9.3. GET /api/usuarios/{usuarioId}/semanas/{semanaId}/subbloques

9.4. GET /api/usuarios/{usuarioId}/subbloques/{subBloqueId}/ejercicios-sugeridos

11. Series de entrenamiento de Usuario

11.1. GET /api/usuarios/{usuarioId}/ejercicios/{ejercicioId}/series

11.2. POST /api/usuarios/{usuarioId}/ejercicios/{ejercicioId}/series

11.3. PUT /api/usuarios/{usuarioId}/series/{serieId}

11.4. DELETE /api/usuarios/{usuarioId}/series/{serieId}

12. Gráficas / Progreso

12.1. GET /api/grafica/{usuarioId}/ejercicios

12.2. GET /api/grafica/{usuarioId}/ejercicios/{ejercicioUsuarioId}/max-pesos

13. Manejo de errores y respuestas comunes

14. DTOs usados en requests/responses

15. Seguridad y JwtFilter

16. Notas de despliegue del backend

17. Diagrama ER de la base de datos

1. Introducción
   El backend de FitTrack ofrece una API REST construida con Spring Boot para gestionar toda la lógica y persistencia de datos relacionada con la planificación y seguimiento de entrenamientos. Su propósito es servir de capa intermedia entre la base de datos relacional (por ejemplo, MySQL) y el frontend (web o móvil), de modo que maneje autenticación, autorización, reglas de negocio, validaciones y exposición de recursos en formato JSON.

   En líneas generales, el sistema soporta dos roles principales:

- **Entrenador/Admin**: puede crear y gestionar bloques de entrenamiento, semanas, sub-bloques, ejercicios sugeridos y sus series asociadas; asignar o desasignar bloques a usuarios; consultar el conjunto de usuarios a su cargo.

- **Usuario**: accede a los bloques asignados; recorre semanas y sub-bloques; consulta ejercicios sugeridos; agrega ejercicios personales; registra series de entrenamiento con detalles (peso, repeticiones, fecha, esfuerzo); consultas métricas como peso máximo por fecha; añade notas de calendario si aplica.
  Puntos clave de la arquitectura
- **Spring Boot + Spring Security + JWT**:
  - Se emplea JWT sin estado para cada petición, validado en un filtro (JwtFilter) y configurado en SecurityConfig.

    - Rutas públicas limitadas a login y registro; el resto requiere encabezado Authorization: Bearer <token>, con permisos según rol.
- **Capas bien separadas**:
    - **Controladores** (@RestController): exponen endpoints, validan presencia/formato de token, extraen parámetros y delegan en servicios.
    - **Servicios**: contienen la lógica de negocio y validaciones (existencia de recursos, pertenencia de recursos al usuario correcto, integridad de datos) y usan repositorios para operaciones en BD, devolviendo DTOs.
    - **Repositorios**: Spring Data JPA para interactuar con las entidades JPA que representan tablas (Roles, Usuarios, Entrenadores, Bloques, Semanas, SubBloques, EjerciciosSugeridos, SeriesSugeridas, EjerciciosUsuario, SeriesEntrenamiento, NotasCalendario, Tokens, etc.).
    - **DTOs**: desacoplan entidades de la superficie JSON, definiendo exactamente qué campos se envían o reciben.
- **Validaciones y manejo de errores**:
    - Servicios lanzan IllegalArgumentException (o similares) con mensajes claros (“Recurso no encontrado”, “Acceso denegado”, “Token inválido”), y los controladores devuelven códigos HTTP adecuados: 400, 401, 403, 404 según el caso.
    - Se evita exponer detalles internos o stack traces en las respuestas.
- **Consistencia de rutas**:
    - Rutas agrupadas por funcionalidad y rol, usando convenciones REST:
        - /api/auth/login, /api/registro → autenticación y registro de usuarios.
        - /api/admin/... → operaciones de entrenadores/admin (CRUD de bloques, semanas, sub-bloques, ejercicios sugeridos/series sugeridas, asignaciones).

    - /api/usuarios/... → operaciones propias del usuario (ver bloques/semanas/sub-bloques asignados, gestionar ejercicios y series).
    - /api/grafica/... → métricas/puntos de datos (peso máximo por fecha, lista de ejercicios).
- **Formato de datos**: JSON en cuerpo de petición y respuesta; fechas en ISO (YYYY-MM-DD) para LocalDate, arrays o mapas cuando convenga; respuestas exitosas con objetos o mensajes breves; errores con texto o JSON con mensaje.
- **CORS**: configurado para permitir orígenes de frontend (p. ej., http://localhost:8100) evitando problemas en desarrollo.

2. Autenticación y Registro

2.1 POST /api/auth/login

Ruta: /api/auth/login

Método HTTP: POST

Descripción: Autentica credenciales de un usuario o entrenador, y devuelve un JWT junto con información básica (id, correo, nombre y roles) para usar en futuras peticiones.

Seguridad / Acceso: Público (no se requiere token).

Headers:

Content-Type: application/json

Request Body: JSON con las credenciales.

```
{
  "correo": "usuario@example.com",
  "contrasena": "textoPlano"
}
```

correo (String, obligatorio): correo electrónico del usuario o entrenador.

contrasena (String, obligatorio): contraseña en texto plano que se compara con la almacenada en la base de datos (codificada con BCrypt).

Flujo interno:

1. Búsqueda de usuario en tabla usuario:

- Se invoca usuarioRepository.findByCorreo(correo).
- Si existe:
  - Se obtiene la entidad Usuario, se compara la contraseña recibida con la contraseña almacenada (hash BCrypt) usando PasswordEncryptor.checkPassword(raw, hashed).
  - Si no coincide, se responde con 401.
  - Si coincide, se genera un JWT con jwtUtil.create(email, rolNombre), donde:
    - El token incluye como subject el email.
    - Se añade claim "password" con el nombre del rol (por ejemplo, "usuario").
    - Expiración a 7 días.
  - Se construye respuesta con un Map que contiene:
    - token: el JWT.
    - id: id del usuario.
    - correo: correo del usuario.
    - nombre: nombre del usuario.
    - roles: array con el nombre del rol (p. ej. ["usuario"]).

2. **Si no existe en usuario, se busca en tabla entrenador**:
   - entrenadorRepository.findByCorreo(correo).
   - Si existe:
     - Similar comparación de contraseña con PasswordEncryptor.checkPassword.
     - Si coincide, se genera JWT con jwtUtil.create(email, "admin") (o rol asociado al entrenador, en código se fija "admin").
     - Se responde con un Map con:

        - token: JWT.
        - id: id del entrenador.
        - correo: correo del entrenador.
        - nombre: nombre del entrenador.
        - roles: ["admin"].
    - Si no coincide o no existe, se devuelve 404 o 401:
        - Si no existe en ambas tablas: retorna 404 Not Found con cuerpo "Usuario o entrenador no encontrado".
        - Si existe, pero contraseña errónea: retorna 401 Unauthorized con cuerpo "Contraseña incorrecta".

Respuestas:

- **200 OK** (autenticación exitosa):

```
{
  "token": "eyJhbGciOiJIUzI1NiJ9....",
  "id": 5,
  "correo": "usuario@example.com",
  "nombre": "Nombre Usuario",
  "roles": ["usuario"]
}
```

    - El campo token se usará en encabezado Authorization: Bearer <token> para siguientes peticiones.
- **401 Unauthorized**:
    - Cuando falta o no se envía JSON válido, o la contraseña no coincide:
        - Cuerpo: "Contraseña incorrecta" (si el correo existe pero contraseña errónea).
        - O bien: si en código detecta ausencia de token no aplica aquí, pero si JSON mal formado, Spring devolverá 400.
- **404 Not Found**:
    - Si ni usuario ni entrenador con ese correo existen: cuerpo "Usuario o entrenador no encontrado".
- **500 Internal Server Error**:
    - Si ocurre un error inesperado (por ejemplo, problemas de conexión a BD). Es recomendable capturar excepciones

específicas o usar un manejador global de excepciones para devolver mensajes controlados.

Ejemplo de uso con curl:

```
curl -X POST http://localhost:8080/api/auth/login \
  -H "Content-Type: application/json" \
  -d '{"correo":"adrian@gmail.com","contrasena":"password123"}'
```

nombre (String, obligatorio): nombre legible del usuario.

- correo (String, obligatorio): correo único; si ya existe un usuario con ese correo, se lanzará excepción.
- contrasena (String, obligatorio): contraseña en texto plano; se codificará con BCrypt antes de guardarla.

Flujo interno:

1. Verificar si existe ya un usuario con el mismo correo: usuarioRepository.existsByCorreo(correo).
    - Si existe, el servicio lanza IllegalStateException("Ya existe un usuario registrado con ese correo.").
    - Actualmente, la excepción no está capturada en el controlador, por lo que propagará un error 500 si no se maneja globalmente. Se recomienda manejarla para devolver 400 o 409.
2. Obtener el rol “usuario” desde rolRepository.findByNombre("usuario").
    - Si no existe ese rol en la base de datos, lanza IllegalStateException.
3. Crear entidad Usuario:
    - Asignar nombre, correo.

- Codificar contraseña con passwordEncoder.encode(contrasena).
- Asignar rolUsuario.
- entrenador queda nulo inicialmente.

4. Guardar en BD: usuarioRepository.save(nuevoUsuario).
5. Construir DTO de respuesta UsuarioDTO, con:
   - id generado.
   - nombre, correo, rol, entrenadorId (nulo).
6. Devolver 200 OK con el DTO en JSON.

Respuestas:

- **200 OK**:

-

400 Bad Request o **409 Conflict** (recomendado):

- Si el correo ya está registrado, el servicio lanza excepción. Idealmente, se debería capturar y devolver:
  - Código: 400 o 409.
  - Cuerpo: "Ya existe un usuario registrado con ese correo."
- En la implementación actual, sin manejo global, se obtendría un 500 Internal Server Error. Se sugiere añadir un controlador

de excepciones para interceptar IllegalStateException y devolver 400 con mensaje claro.

- **500 Internal Server Error**:
  - Si fallo inesperado (por ejemplo, la tabla roles no contiene “usuario”).
  - Si no se controla la excepción del correo duplicado, resultará en 500: es importante mejorar el manejador global.

Ejemplo de uso con curl:

```
curl -X POST http://localhost:8080/api/registro \
  -H "Content-Type: application/json" \
  -d '{"nombre":"María Pérez","correo":"maria@example.com","contrasena":"miSecreta"}'
```

Si el correo no existía, se devolverá el JSON del nuevo usuario.

Si ya existe, conviene que la API devuelva 400 con mensaje de “Ya existe un usuario registrado con ese correo.”

3. Gestión de Usuarios y Entrenadores (ADMIN)

3.1 GET /api/admin/usuarios

Ruta: /api/admin/usuarios

Método HTTP: GET

Descripción: Devuelve la lista de todos los usuarios registrados en el sistema. Está pensado para administradores (entrenadores con rol “admin”).

Seguridad / Acceso:

- Requiere token JWT en el header Authorization: Bearer <token>.
- El token debe corresponder a un entrenador cuyo rol sea “admin”.

Headers:

- Authorization: Bearer <token>

Request Body: Ninguno.

Flujo interno:

1. **Leer encabezado Authorization**:
    - Si no existe o no empieza por “Bearer ”, se responde 401 Unauthorized (sin cuerpo o con mensaje “Token no proporcionado o inválido”).
2. **Validar el token**:
    - jwtUtil.validateToken(token). Si inválido o expirado, se responde 401 Unauthorized con mensaje “Token inválido o expirado”.
3. **Obtener entrenador desde token**:
    - En el controlador:

```
Entrenador entrenador = entrenadorService.obtenerEntrenadorDesdeToken(token);
```

Si no se encuentra un entrenador con el correo extraído del token, o el token es inválido, se lanzaría IllegalArgumentException dentro del servicio, capturado aquí devolviendo 401 Unauthorized.

Verificar rol “admin”:

- Se comprueba entrenador.getRol().getNombre() es igual a “admin” (ignorando mayúsculas/minúsculas). Si no es admin, se responde 403 Forbidden.

Llamar al servicio para obtener usuarios:

- List<UsuarioDTO> usuarios = usuarioService.obtenerTodosUsuarios();
- Este método recorre todos los registros de Usuario en la BD y los mapea a UsuarioDTO, que incluye: id, nombre, correo, rol, entrenadorId.

Devolver respuesta:

- 200 OK con cuerpo JSON: lista de objetos UsuarioDTO. Ejemplo:

```
[
  {
    "id": 1,
    "nombre": "Adrián",
    "correo": "adrian@gmail.com",
    "rol": "usuario",
    "entrenadorId": 1
  },
  {
    "id": 2,
    "nombre": "Chema",
    "correo": "chema@gmail.com",
    "rol": "usuario",
    "entrenadorId": 1
  },
  ...
]
```

Respuestas posibles:

- **200 OK**: Lista de usuarios.
- **401 Unauthorized**:
  - Si falta header Authorization o no empieza por Bearer.
  - Si el token es inválido o expirado.
  - Si con el token no se encuentra un entrenador en la BD.
- **403 Forbidden**:
  - El token es válido pero el entrenador no tiene rol “admin”.
- **500 Internal Server Error**:
  - Si ocurre un error inesperado (por ejemplo, error de conexión a BD). Es recomendable tener un manejador global de excepciones para devolver mensajes controlados.

Ejemplo de uso con curl:

```
curl -X GET http://localhost:8080/api/admin/usuarios \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiJ9..."
```

Si el token pertenece a un entrenador admin válido, se recibe la lista de usuarios.

3.2 POST /api/admin/usuarios/{usuarioId}/asignar-entrenador-auto/{entrenadorId}

Ruta: /api/admin/usuarios/{usuarioId}/asignar-entrenador-auto/{entrenadorId}

Método HTTP: POST

Descripción: Asigna automáticamente al usuario identificado por usuarioId el entrenador identificado por entrenadorId. Está diseñado para usarse tras el registro de un usuario para asignarle un entrenador por defecto sin requerir token de administrador.

Seguridad / Acceso:

- **Actualmente, es público**: no se valida token.
- **Atención**: esto implica que cualquiera que conozca IDs puede invocar esta ruta para asignar un entrenador a un usuario; conviene revisarlo si se requiere mayor control. Según diseño original, se permite sin token para facilitar asignación en flujo de registro automático.

Headers:

- Ninguno obligatorio.

Request Body: Ninguno.

Path Parameters:

- usuarioId (Long): ID de usuario a asignar.
- entrenadorId (Long): ID de entrenador a asignar al usuario.

Flujo interno:

1. **No se lee ni valida token** en el controlador. Simplemente se llama al servicio:

```
usuarioService.asignarEntrenadorAUsuario(usuarioId, entrenadorId);
```

Dentro de UsuarioServiceImpl.asignarEntrenadorAUsuario:

   - Verificar existencia del usuario: usuarioRepository.findById(usuarioId). Si no existe: lanza IllegalArgumentException("Usuario no encontrado: " + usuarioId).
   - Verificar existencia del entrenador: entrenadorRepository.findById(entrenadorId). Si no existe: lanza IllegalArgumentException("Entrenador no encontrado: " + entrenadorId).
   - Ejecutar update en BD: usuarioRepository.asignarEntrenadorAUsuario(usuarioId, entrenadorId). Si devuelve 0 filas actualizadas: lanza IllegalArgumentException("No se pudo asignar entrenador al usuario").

2. **Devolver respuesta**:
   - En controlador, ante finalización exitosa, responde 204 No Content.
   - Si se lanza IllegalArgumentException, interrumpe y retorna 500 si no se captura. En la implementación de EntrenadorController, sí envuelven la llamada en try/catch para devolver 400 Bad Request con el mensaje de la excepción; pero en este endpoint específico no hay try/catch: devolviendo 204 o en error lanzará 500. Se recomienda añadir

manejo para devolver 400 Bad Request cuando la excepción es IllegalArgumentException.

3. Respuestas posibles:

- **204 No Content**: Asignación completada con éxito.
- **400 Bad Request** (recomendado si se captura IllegalArgumentException):
    - Mensaje en cuerpo: texto de la excepción ("Usuario no encontrado: X" o "Entrenador no encontrado: Y" o "No se pudo asignar entrenador al usuario").
- **500 Internal Server Error**:
    - Si la excepción no se captura localmente. Es aconsejable envolver en try/catch en el controlador o usar @ControllerAdvice.

Ejemplo de uso con curl:

```
curl -X POST http://localhost:8080/api/admin/usuarios/5/asignar-entrenador-auto/2
```

- Si el usuario con ID 5 existe y el entrenador con ID 2 existe, la petición devuelve 204 sin cuerpo.

3.3 GET /api/admin/entrenadores/{entrenadorId}/usuarios

Ruta: /api/admin/entrenadores/{entrenadorId}/usuarios

Método HTTP: GET

Descripción: Obtiene la información del entrenador identificado por entrenadorId, incluyendo el conjunto de usuarios asignados a ese entrenador. Está destinado a que cada entrenador "admin" vea sus propios usuarios.

Seguridad / Acceso:

- Requiere token JWT en header Authorization: Bearer <token>.
- El token debe corresponder a un entrenador.
- Además, el ID extraído del token debe coincidir con entrenadorId en la ruta; es decir, un entrenador sólo puede consultar sus propios usuarios.

Headers:

- Authorization: Bearer <token>

Request Body: Ninguno.

Path Parameters:

- entrenadorId (Long): ID del entrenador cuyas relaciones de usuarios queremos obtener.

Flujo interno:

1. **Leer encabezado Authorization**:
    - Si falta o no es Bearer: 401 Unauthorized.
2. **Validar token**:
    - jwtUtil.validateToken(token). Si inválido/expirado: 401 Unauthorized.
3. **Obtener entrenador autenticado**:
    - Entrenador entrenador = entrenadorService.obtenerEntrenadorDesdeToken(token);
    - Si no existe o token inválido: lanza IllegalArgumentException, capturar y enviar 401 Unauthorized.
4. **Comparar IDs**:
    - Si entrenador.getId() no coincide con el entrenadorId de la ruta: 403 Forbidden (“No puedes acceder a los datos de otro entrenador.”).

5. **Llamar al servicio para obtener datos**:
    - EntrenadorDTO dto = entrenadorService.obtenerEntrenadorConUsuarios(entrenadorId);
    - Este método hace:
        1. Buscar la entidad Entrenador por ID; si no existe, lanza IllegalArgumentException("Entrenador no encontrado con ID: " + entrenadorId).
        2. Consultar todos los usuarios asignados: entrenadorRepository.findUsuariosByEntrenadorId(entrenadorId), obteniendo Set<Usuario>.
        3. Mapear cada Usuario a UsuarioDTO(u, true), donde el constructor con sinEntrenadorId=true incluye datos id, nombre, correo, rol, pero sin incluir entrenadorId en el DTO (según implementación de UsuarioDTO(Usuario, boolean)).
        4. Construir EntrenadorDTO con id, nombre, correo, rol, y setUsuarios.
6. **Devolver respuesta**:
    - 200 OK con cuerpo JSON representando EntrenadorDTO. Ejemplo:

```
{
  "id": 1,
  "nombre": "Sergio",
  "correo": "sergio@gmail.com",
  "rol": "admin",
  "usuarios": [
    {
      "id": 1,
      "nombre": "Adrián",
      "correo": "adrian@gmail.com",
      "rol": "usuario"
    },
    {
      "id": 2,
      "nombre": "Chema",
      "correo": "chema@gmail.com",
      "rol": "usuario"
    }
  ]
}
```


Respuestas posibles:

- **200 OK**: JSON con EntrenadorDTO y su lista de usuarios.
- **401 Unauthorized**:
    - Falta o mal header Authorization.
    - Token inválido o expirado.
    - Token no corresponde a un entrenador existente.
- **403 Forbidden**:
    - El token es válido pero el ID extraído no coincide con entrenadorId en la ruta.
- **404 Not Found**:
    - Si entrenadorService.obtenerEntrenadorConUsuarios lanza IllegalArgumentException("Entrenador no encontrado..."). En

el controlador se captura y devuelve 404 Not Found con el mensaje.

- **500 Internal Server Error**:
    - Errores inesperados. Recomendable controlador global de excepciones.

Ejemplo de uso con curl:

```
curl -X GET http://localhost:8080/api/admin/entrenadores/1/usuarios \
  -H "Authorization: Bearer eyJhbGciOiJIUzI1NiJ9..."
```

Si el token pertenece al entrenador con ID 1, se recibe su DTO con lista de usuarios.

- Si el token pertenece a entrenador distinto, se recibe 403 Forbidden.

4. Gestión de Bloques (Entrenador/Admin)

4.1 POST /api/admin/bloques

Ruta: /api/admin/bloques
Método HTTP: POST
Descripción: Crea un nuevo bloque asociado al entrenador que hace la petición. Opcionalmente se puede indicar un usuarioId para asignar el bloque a un usuario en el momento de la creación.
Seguridad / Acceso:

- Requiere token JWT en header Authorization: Bearer <token>.
- El token debe corresponder a un entrenador (cualquier rol de entrenador; en general, entrenadores tienen permiso para crear bloques; no se exige rol “admin” a nivel de Spring Security, pero en la lógica se valida que exista un entrenador con ese correo).

Headers:

- Authorization: Bearer <token>
- Content-Type: application/json

Request Body (JSON):

```
{
  "nombre": "Nombre del bloque",
  "usuarioId": 5   // opcional, si se desea asignar directamente a un usuario
}
```

nombre (String, obligatorio): Nombre descriptivo del bloque.

- usuarioId (Long, opcional): ID de un usuario existente que pertenezca al entrenador. Si no se incluye o es null, el bloque se crea sin asignar a ningún usuario (usuario = null).

Flujo interno:

1. **Leer encabezado Authorization**:
    - Si falta o no empieza por “Bearer ”, devuelve 401 Unauthorized con mensaje “Token no proporcionado o inválido.”.
2. **Validar token**:
    - jwtUtil.validateToken(token). Si inválido o expirado, devuelve 401 Unauthorized con mensaje “Token inválido o expirado.”.
3. **Obtener correo del entrenador**:
    - String correo = jwtUtil.extractEmail(token).
4. **Verificar que el correo corresponde a un entrenador**:
    - entrenadorRepository.findByCorreo(correo). Si no se encuentra, devuelve 403 Forbidden con mensaje “Acceso denegado. Solo entrenadores pueden crear bloques.”.
5. **Llamar al servicio**:
    - Bloque bloque = bloqueService.crearBloque(requestDTO, entrenadorId).
    - En BloqueServiceImpl.crearBloque:
        1. Se busca la entidad Entrenador por el entrenadorId; si no existe, lanza IllegalArgumentException("Entrenador no encontrado.").
        2. Se crea instancia Bloque y se asigna nombre y entrenador.
        3. Si requestDTO.getUsuarioId() no es null:
            - Se busca Usuario por ese ID; si no existe, lanza IllegalArgumentException("Usuario no encontrado.").

        - Se asigna bloque.setUsuario(usuario).
    4. Se guarda en repositorio bloqueRepository.save(bloque).
6. **Devolver respuesta**:
    - Si ha salido bien, responde 201 Created con cuerpo JSON:

```
{
  "mensaje": "Bloque creado correctamente",
  "bloqueId": 123
}
```

    - Si IllegalArgumentException durante creación (entrenador no encontrado, usuario no existe, etc.), se captura en el controlador y devuelve 400 Bad Request con el mensaje de la excepción.
    - Otros errores: 500 Internal Server Error si no se capturan.

 Respuestas posibles:

- **201 Created**:

```
{
  "mensaje": "Bloque creado correctamente",
  "bloqueId": 123
}
```

400 Bad Request:

- Ejemplos de mensajes:
    - “Entrenador no encontrado.”
    - “Usuario no encontrado.”
- **401 Unauthorized**:
    - Token faltante o no empieza por Bearer.
    - Token inválido o expirado.
- **403 Forbidden**:
    - El token es válido pero no corresponde a un entrenador en la BD.
- **500 Internal Server Error**:
    - Errores inesperados.

Ejemplo con curl:

```
curl -X POST http://localhost:8080/api/admin/bloques \
  -H "Authorization: Bearer <token_entrenador>" \
  -H "Content-Type: application/json" \
  -d '{"nombre":"Fuerza Básica","usuarioId":5}'
```

4.2 GET /api/admin/bloques

Ruta: /api/admin/bloques

Método HTTP: GET

Descripción: Obtiene la lista de bloques creados por el entrenador autenticado. Devuelve un conjunto de BloqueDTO, que incluyen id, nombre, posible usuario asignado (id y nombre) y entrenadorId.

Seguridad / Acceso:

- Requiere token JWT en header Authorization: Bearer <token>.
- El token debe corresponder a un entrenador en la BD.

Headers:

- Authorization: Bearer <token>

Request Body: Ninguno.

Flujo interno:

1. **Leer encabezado Authorization**:
    - Si falta o no es Bearer, devuelve 401 Unauthorized.
2. **Validar token**:
    - jwtUtil.validateToken(token). Si inválido o expirado, 401 Unauthorized.
3. **Obtener correo y verificar entrenador**:
    - String correo = jwtUtil.extractEmail(token).
    - entrenadorRepository.findByCorreo(correo). Si no existe, 403 Forbidden.

4. **Llamar al servicio**:
    - Set<BloqueDTO> bloques = bloqueService.listarBloquesPorEntrenador(entrenadorId).
    - En BloqueServiceImpl.listarBloquesPorEntrenador:
        1. bloqueRepository.findByEntrenadorId(entrenadorId): obtiene todos los bloques donde entrenador_id = entrenadorId.
        2. Para cada Bloque b, se crea UsuarioMiniDTO si b.getUsuario() != null, con id y nombre.
        3. Se construye new BloqueDTO(b.getId(), b.getNombre(), usuarioDto, b.getEntrenador().getId()).
        4. Se recolecta en un Set<BloqueDTO>.
5. **Devolver respuesta**:
    - 200 OK con cuerpo JSON: lista (array) de objetos BloqueDTO. Ejemplo:


```
[
  {
    "id": 10,
    "nombre": "Fuerza Básica",
    "usuario": { "id": 5, "nombre": "Adrián" },
    "entrenadorId": 1
  },
  {
    "id": 11,
    "nombre": "Resistencia Cardio",
    "usuario": null,
    "entrenadorId": 1
  }
]
```


Respuestas posibles:

- **200 OK**: Lista de bloques (vacía si no hay).

- **401 Unauthorized**: Token faltante o inválido.
- **403 Forbidden**: Token válido, pero no corresponde a un entrenador en BD.
- **500 Internal Server Error**: Errores imprevistos.

Ejemplo con curl:

```
curl -X GET http://localhost:8080/api/admin/bloques \
  -H "Authorization: Bearer <token_entrenador>"
```

4.3 PUT /api/admin/bloques/{bloqueId}

Ruta: /api/admin/bloques/{bloqueId}

Método HTTP: PUT

Descripción: Actualiza un bloque existente (nombre y usuario asignado) siempre que el bloque pertenezca al entrenador autenticado.

Seguridad / Acceso:

- Requiere token JWT en header Authorization: Bearer <token>.
- El token debe corresponder a un entrenador en BD.

Headers:

- Authorization: Bearer <token>
- Content-Type: application/json

Path Parameters:

- bloqueId (Long): ID del bloque a actualizar.

Request Body (JSON):

```
{
  "nombre": "Nuevo nombre",
  "usuarioId": 7
}
```

nombre (String, obligatorio): Nombre nuevo del bloque.

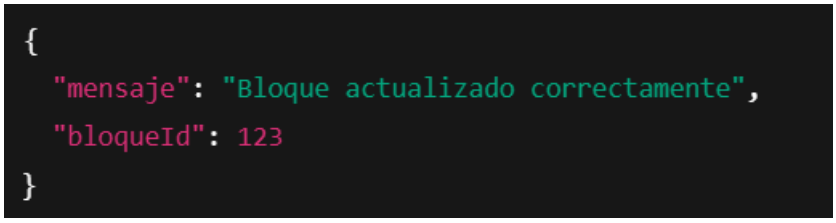
- usuarioId (Long, obligatorio): ID de usuario al que se asigna el bloque. Debe existir y pertenecer al entrenador (la verificación de pertenencia se hace en otro endpoint, pero aquí se asume que el servicio lanzará excepción si no existe).

Flujo interno:

1. **Leer y validar header Authorization**:
    - Si falta o malformado: 401 Unauthorized.
2. **Validar token**:
    - jwtUtil.validateToken(token). Si inválido/expirado: 401 Unauthorized.
3. **Obtener correo y verificar entrenador**:
    - String correo = jwtUtil.extractEmail(token).
    - entrenadorRepository.findByCorreo(correo). Si no existe: 403 Forbidden.
    - Obtener entrenadorId para la comparación.
4. **Llamar al servicio**:
    - Bloque actualizado = bloqueService.actualizarBloque(bloqueId, requestDTO, entrenadorId).
    - En BloqueServiceImpl.actualizarBloque:
        1. bloqueRepository.findByIdAndEntrenadorId(bloqueId, entrenadorId). Si no encuentra (no existe o pertenece a otro entrenador): lanza IllegalArgumentException("Bloque no encontrado o no pertenece al entrenador: " + bloqueId).
        2. usuarioRepository.findById(requestDTO.getUsuarioId()). Si no existe: lanza IllegalArgumentException("Usuario no encontrado: " + usuarioId).

3. Actualizar campos: bloque.setNombre(...), bloque.setUsuario(usuario).
4. Guardar: bloqueRepository.save(bloque).

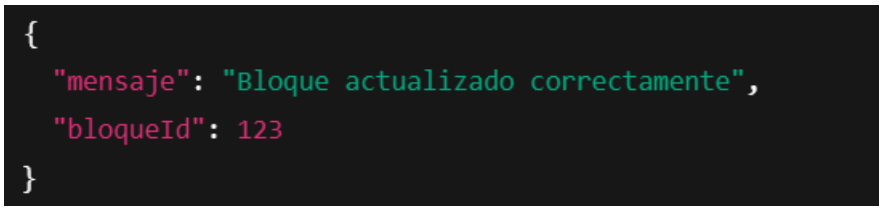
5. **Devolver respuesta**:
   - Si todo OK: 200 OK con cuerpo JSON:



   - Si IllegalArgumentException: se captura y devuelve 404 Not Found (según implementación) con el mensaje. Algunas implementaciones podrían devolver 400 Bad Request, pero el controlador usa status(HttpStatus.NOT_FOUND) en catch, devolviendo 404.
   - Otros errores: 500 Internal Server Error.

Respuestas posibles:

- **200 OK**:



- **401 Unauthorized**: Token ausente o inválido.
- **403 Forbidden**: Token válido pero no corresponde a un entrenador en BD.
- **404 Not Found**:
  - Bloque no existe o no pertenece al entrenador.
  - Usuario (usuarioId) no existe.

- **500 Internal Server Error**: Errores imprevistos.

Ejemplo con curl:

```
curl -X PUT http://localhost:8080/api/admin/bloques/123 \
  -H "Authorization: Bearer <token_entrenador>" \
  -H "Content-Type: application/json" \
  -d '{"nombre":"Fuerza Avanzada","usuarioId":5}'
```

4.4 DELETE /api/admin/bloques/{bloqueId}

Ruta: /api/admin/bloques/{bloqueId}

Método HTTP: DELETE

Descripción: Elimina un bloque existente siempre que pertenezca al entrenador autenticado.

Seguridad / Acceso:

- Requiere token JWT en header Authorization: Bearer <token>.
- El token debe corresponder a un entrenador en BD.

Headers:

- Authorization: Bearer <token>

Path Parameters:

- bloqueId (Long): ID del bloque a eliminar.

Request Body: Ninguno.

Flujo interno:

1. **Leer y validar token**:
    - Igual que en PUT: si falta o inválido → 401.
2. **Obtener correo y verificar entrenador**:
    - String correo = jwtUtil.extractEmail(token).

- entrenadorRepository.findByCorreo(correo). Si no existe: 403 Forbidden.
- Long entrenadorId.

3. **Llamar al servicio**:
   - bloqueService.eliminarBloque(bloqueId, entrenadorId).
   - En BloqueServiceImpl.eliminarBloque:
     1. bloqueRepository.findByIdAndEntrenadorId(bloqueId, entrenadorId). Si no existe o no pertenece: lanza IllegalArgumentException("Bloque no encontrado o no pertenece al entrenador: " + bloqueId).
     2. bloqueRepository.delete(bloque).
4. **Devolver respuesta**:
   - Si OK: 200 OK con cuerpo JSON:

```
{
  "mensaje": "Bloque eliminado correctamente",
  "bloqueId": 123
}
```

   - Si IllegalArgumentException: se captura y devuelve 404 Not Found con mensaje.
   - Otros errores: 500 Internal Server Error.

Respuestas posibles:

- **200 OK**:

```
{
  "mensaje": "Bloque eliminado correctamente",
  "bloqueId": 123
}
```

- **401 Unauthorized**: Token ausente o inválido.
- **403 Forbidden**: Token válido pero no corresponde a entrenador en BD.

- **404 Not Found**: Bloque no existe o no pertenece al entrenador.
- **500 Internal Server Error**: Errores imprevistos.

Ejemplo con curl:

```
curl -X DELETE http://localhost:8080/api/admin/bloques/123 \
  -H "Authorization: Bearer <token entrenador>"
```

4.5 POST /api/admin/bloques/{bloqueId}/asignar/{usuarioId}

Ruta: /api/admin/bloques/{bloqueId}/asignar/{usuarioId}

Método HTTP: POST

Descripción: Asigna un bloque existente al usuario indicado, siempre que:

1. El bloque exista y pertenezca al entrenador autenticado.
2. El usuario exista y esté asignado a ese mismo entrenador.

Seguridad / Acceso:

- Requiere token JWT en header Authorization: Bearer <token>.
- El token debe corresponder a un entrenador en BD, y ese entrenador debe tener rol “admin” según la lógica del controlador (aunque el término “admin” aquí se refiere al rol del entrenador, no al path de Spring Security).

Headers:

- Authorization: Bearer <token>

Path Parameters:

- bloqueId (Long): ID del bloque a asignar.
- usuarioId (Long): ID del usuario al que se asigna el bloque.

Request Body: Ninguno.

Flujo interno:

1. **Leer y validar token**:
    - Si falta o no es Bearer → 401 Unauthorized.
    - Extraer token limpio: token.substring(7).
    - if (!jwtUtil.validateToken(token)) → 401 Unauthorized.
2. **Obtener entrenador desde token y verificar rol "admin"**:
    - Entrenador entrenador = entrenadorService.obtenerEntrenadorDesdeToken(cleanToken).
        - En este método se extrae email, valida token, busca entrenador en BD, lanza IllegalArgumentException si no existe o token inválido.
    - Comprobar entrenador.getRol().getNombre().equalsIgnoreCase("admin"). Si no es admin: 403 Forbidden con mensaje "Acceso denegado. Se requiere rol admin.".
3. **Verificar existencia y pertenencia del usuario**:
    - Usuario usuarioDestino = usuarioService.obtenerUsuarioPorIdEntidad(usuarioId).orElseThrow(...).
        - Si no existe → lanza IllegalArgumentException("Usuario destino no encontrado."), atrapado en controlador y devuelve 400 Bad Request.
    - Verificar que usuarioDestino.getEntrenador() != null y usuarioDestino.getEntrenador().getId().equals(entrenador.getId()). Si no, 403 Forbidden con mensaje "El usuario no pertenece a este entrenador.".
4. **Llamar al servicio para asignar**:

- bloqueService.asignarBloqueAUsuario(bloqueId, usuarioId).
- En BloqueServiceImpl.asignarBloqueAUsuario:

1. bloqueRepository.findById(bloqueId). Si no existe: lanza IllegalArgumentException("Bloque no encontrado").

2. usuarioRepository.findById(usuarioId). Aunque ya verificado, se busca de nuevo: si no existe, lanza excepción.

3. Verificar si ya está asignado a ese usuario: si bloque.getUsuario() != null && bloque.getUsuario().getId().equals(usuarioId), lanza IllegalArgumentException("El bloque ya está asignado a este usuario.").

4. bloque.setUsuario(usuario) y bloqueRepository.save(bloque).

5. **Devolver respuesta**:
   - 200 OK con cuerpo JSON:

```
{
  "mensaje": "Bloque asignado correctamente al usuario"
}
```

   - Si IllegalArgumentException en la lógica (bloque no existe, usuario no existe, ya asignado, etc.), se captura y devuelve 400 Bad Request con el mensaje.
   - Si excepción de tipo diferente o no prevista: 500 Internal Server Error.

Respuestas posibles:

- **200 OK**:

```
{
  "mensaje": "Bloque asignado correctamente al usuario"
}
```

400 Bad Request:

- “Usuario destino no encontrado.”
- “Bloque no encontrado”

    - “El bloque ya está asignado a este usuario.”
- **401 Unauthorized**:
    - Token faltante o inválido o expirado.
- **403 Forbidden**:
    - Token válido pero el entrenador no tiene rol “admin”.
    - Usuario no pertenece a este entrenador.
- **500 Internal Server Error**: Errores imprevistos.

Ejemplo con curl:

```
curl -X POST http://localhost:8080/api/admin/bloques/123/asignar/5 \
  -H "Authorization: Bearer <token_entrenador_admin>"
```

4.6 DELETE /api/admin/bloques/{bloqueId}/desasignar/{usuarioId}

Ruta: /api/admin/bloques/{bloqueId}/desasignar/{usuarioId}

Método HTTP: DELETE

Descripción: Desasigna (quita) el bloque del usuario indicado, dejando usuario_id = null en la entidad Bloque. Solo si el bloque existe y está asignado a ese usuario, y el usuario pertenece al entrenador autenticado, y el bloque pertenece al entrenador.

Seguridad / Acceso:

- Requiere token JWT en header Authorization: Bearer <token>.
- El token debe corresponder a un entrenador con rol “admin”.

Headers:

- Authorization: Bearer <token>

Path Parameters:

- bloqueId (Long): ID del bloque a desasignar.
- usuarioId (Long): ID del usuario del que se quita el bloque.

Request Body: Ninguno.

Flujo interno:

1. **Leer y validar token**:
    - Si falta o no es Bearer → 401 Unauthorized.
    - String token = header.substring(7), jwtUtil.validateToken(token), si inválido → 401.
2. **Obtener entrenador y verificar rol "admin"**:
    - Entrenador entrenador = entrenadorService.obtenerEntrenadorDesdeToken(token).
    - Si rol no es "admin" → 403 Forbidden.
3. **Verificar existencia y pertenencia del usuario**:
    - Usuario usuarioDestino = usuarioService.obtenerUsuarioPorIdEntidad(usuarioId).orElseThrow(...).
        - Si no existe → IllegalArgumentException("Usuario no encontrado: " + usuarioId), capturado y devuelto 400 Bad Request o 404 Not Found (según implementación).
    - Verificar usuarioDestino.getEntrenador() != null && usuarioDestino.getEntrenador().getId().equals(entrenador.getId()). Si no → 403 Forbidden.
4. **Llamar al servicio**:
    - bloqueService.desasignarBloqueDeUsuario(bloqueId, usuarioId).
    - En BloqueServiceImpl.desasignarBloqueDeUsuario:

1. bloqueRepository.findById(bloqueId). Si no existe → IllegalArgumentException("Bloque no encontrado: " + bloqueId).

2. Verificar bloque.getUsuario() != null && bloque.getUsuario().getId().equals(usuarioId). Si no, lanza IllegalArgumentException("El bloque " + bloqueId + " no está asignado al usuario " + usuarioId).

3. bloque.setUsuario(null) y bloqueRepository.save(bloque).

5. **Devolver respuesta**:
    - 204 No Content o 200 OK. En la implementación actual devuelve 204 No Content (ResponseEntity.noContent().build()).
    - Si IllegalArgumentException, se lanza: no está capturado explícitamente aquí, se propaga y podría traducirse en 500 o capturarse globalmente para devolver 400/404. Se recomienda capturar y devolver 400 Bad Request o 404 Not Found con el mensaje.

Respuestas posibles:

- **204 No Content**: Desasignación realizada con éxito.
- **400 Bad Request** o **404 Not Found**:
    - "Usuario no encontrado: X"
    - "Bloque no encontrado: Y"
    - "El bloque Y no está asignado al usuario X"
- **401 Unauthorized**: Token ausente o inválido.
- **403 Forbidden**: Token válido pero rol no es "admin", o el usuario no pertenece al entrenador.
- **500 Internal Server Error**: Errores imprevistos.

Ejemplo con curl:

```
curl -X DELETE http://localhost:8080/api/admin/bloques/123/desasignar/5 \
  -H "Authorization: Bearer <token_entrenador_admin>"
```

5. Gestión de Semanas (Entrenador)

5.1 POST /api/bloques/{bloqueId}/semanas

Ruta: /api/bloques/{bloqueId}/semanas

Método HTTP: POST

Descripción: Añade una nueva semana al bloque identificado por bloqueId.

Seguridad / Acceso:

- Requiere token JWT en encabezado Authorization: Bearer <token>.
- El token debe corresponder a un entrenador existente en la base de datos.

Headers:

- Authorization: Bearer <token>
- Content-Type: application/json

Path Parameters:

- bloqueId (Long, obligatorio): ID del bloque al que se añade la semana.

Request Body (JSON):

```
{
  "numeroSemana": 3
}
```

numeroSemana (int, obligatorio): Número de la semana dentro del bloque.

Flujo interno:

1. **Leer y validar encabezado Authorization**:

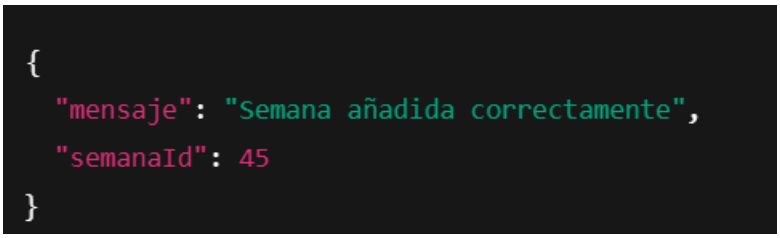
- Si falta o no empieza por “Bearer ”, se responde 401 Unauthorized con mensaje “Token no proporcionado.” o similar.

2. **Validar token**:
    - jwtUtil.validateToken(token). Si inválido o expirado, responde 401 Unauthorized con mensaje “Token inválido o expirado.”.
3. **Obtener email del entrenador**:
    - String correo = jwtUtil.extractEmail(token).
4. **Verificar existencia del entrenador**:
    - entrenadorRepository.findByCorreo(correo). Si no existe, responde 403 Forbidden con mensaje “Acceso denegado. Solo entrenadores.”.
    - Se obtiene entrenadorId para validaciones posteriores.
5. **Llamar al servicio**:

- Semana semana = semanaService.agregarSemana(bloqueId, dto).
- En SemanaServiceImpl.agregarSemana:
    1. Busca Bloque bloque = bloqueRepository.findById(bloqueId). Si no existe, lanza IllegalArgumentException("Bloque no encontrado").
    2. Crea Semana semana = new Semana(), asigna numeroSemana y bloque.
    3. Guarda: semanaRepository.save(semana).

6. Devolver respuesta:

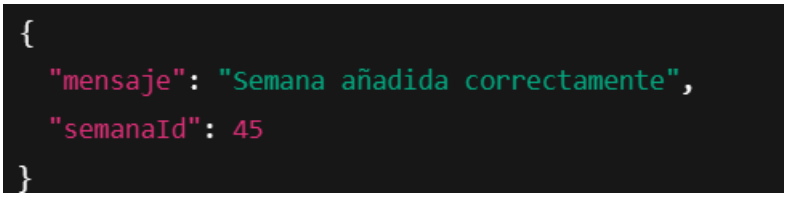
- Si todo OK: 201 Created con cuerpo JSON:



- Si IllegalArgumentException (bloque no existe, o validación extra de pertenencia falla), se captura en el controlador y se devuelve 400 Bad Request con el mensaje de la excepción.
- Otros errores inesperados: 500 Internal Server Error.

Respuestas posibles:

- **201 Created**:



400 Bad Request:

- "Bloque no encontrado"
- "Bloque no pertenece a este entrenador." (si se implementa validación extra)
- Otras validaciones personalizadas (por ejemplo, número de semana duplicado).

- **401 Unauthorized**: Token ausente o inválido.
- **403 Forbidden**: Token válido pero no corresponde a un entrenador en BD.
- **500 Internal Server Error**: Errores imprevistos.

Ejemplo con curl:

```
curl -X POST http://localhost:8080/api/bloques/123/semanas \
  -H "Authorization: Bearer <token_entrenador>" \
  -H "Content-Type: application/json" \
  -d '{"numeroSemana": 3}'
```

5.2 GET /api/bloques/{bloqueId}/semanas

Ruta: /api/bloques/{bloqueId}/semanas

Método HTTP: GET

Descripción: Recupera la lista de semanas asociadas al bloque con ID bloqueId. Devuelve un conjunto de SemanaDTO con id y numeroSemana.

Seguridad / Acceso:

- Requiere token JWT en encabezado Authorization: Bearer <token>.
- El token debe corresponder a un entrenador.

Headers:

- Authorization: Bearer <token>

Path Parameters:

- bloqueId (Long, obligatorio): ID del bloque cuyas semanas se quieren listar.

Request Body: Ninguno.

Flujo interno:

1. **Leer y validar token**:
    - Si falta o inválido → 401 Unauthorized.
2. **Obtener correo y verificar entrenador**:
    - String correo = jwtUtil.extractEmail(token).

- entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden.
- Obtener entrenadorId.

3. **Llamar al servicio**:

- Set<SemanaDTO> semanas = semanaService.obtenerSemanasDeBloque(bloqueId).
- En SemanaServiceImpl.obtenerSemanasDeBloque:
    1. Bloque bloque = bloqueRepository.findById(bloqueId). Si no existe, lanza IllegalArgumentException("Bloque no encontrado").
    2. Recupera Set<Semana> semanas = semanaRepository.findByBloque(bloque).
    3. Mapea a SemanaDTO(s.getId(), s.getNumeroSemana()).
    4. Devuelve Set<SemanaDTO>.
    5. **Devolver respuesta**:
- 200 OK con cuerpo JSON: array de objetos:

```
[
  { "id": 10, "numeroSemana": 1 },
  { "id": 11, "numeroSemana": 2 }
]
```

- Si excepción IllegalArgumentException (bloque no existe o no pertenece): capturar y devolver 404 Not Found o 400 Bad Request con mensaje.
- Otros errores → 500 Internal Server Error.

Respuestas posibles:

- **200 OK**: Lista de semanas (vacía si no hay).
- **400 Bad Request / 404 Not Found**:
    - “Bloque no encontrado”

    - “Bloque no pertenece a este entrenador.” (si se implementa validación extra)

- **401 Unauthorized**: Token ausente o inválido.
- **403 Forbidden**: Token válido pero no encuentra entrenador en BD.
- **500 Internal Server Error**: Errores imprevistos.

Ejemplo con curl:

```
curl -X GET http://localhost:8080/api/bloques/123/semanas \
  -H "Authorization: Bearer <token_entrenador>"
```

5.3 GET /api/semanas/{semanaId}

Ruta: /api/semanas/{semanaId}

Método HTTP: GET

Descripción: Obtiene la información de una semana específica por su ID. Devuelve un objeto SemanaDTO con id y numeroSemana.

Seguridad / Acceso:

- Requiere token JWT en encabezado Authorization: Bearer <token>.
- El token debe corresponder a un entrenador.

Headers:

- Authorization: Bearer <token>

Path Parameters:

- semanaId (Long, obligatorio): ID de la semana a recuperar.

Request Body: Ninguno.

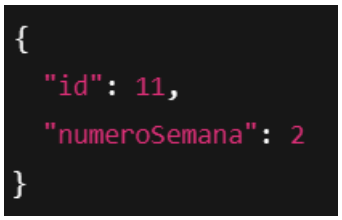
Flujo interno:

1. **Leer y validar token**:
    - Si falta o inválido → 401 Unauthorized.

2. **Obtener y verificar entrenador**:
    - String correo = jwtUtil.extractEmail(token).
    - entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden.
    - Obtener entrenadorId.
3. **Llamar al servicio**:
    - SemanaDTO semana = semanaService.obtenerSemanaPorId(semanaId).
    - En SemanaServiceImpl.obtenerSemanaPorId:
        1. Semana semana = semanaRepository.findById(semanaId). Si no existe, lanza IllegalArgumentException("Semana no encontrada").
        2. Retorna new SemanaDTO(semana.getId(), semana.getNumeroSemana()).

Devolver respuesta:

- Si OK: 200 OK con cuerpo:



    - Si IllegalArgumentException: capturado en controlador y devuelve 404 Not Found o 400 Bad Request según implementación con mensaje “Semana no encontrada” o “Acceso denegado...”.
    - Otros errores → 500 Internal Server Error.

Respuestas posibles:

- **200 OK**:

```
{
  "id": 11,
  "numeroSemana": 2
}
```

404 Not Found / 400 Bad Request:

  - “Semana no encontrada”
  - “Acceso denegado: semana no pertenece a este entrenador.” (si se implementa validación extra)

- **401 Unauthorized**: Token ausente o inválido.
- **403 Forbidden**: Token válido pero no corresponde a un entrenador en BD.
- **500 Internal Server Error**: Errores imprevistos.

Ejemplo con curl:

```
curl -X GET http://localhost:8080/api/semanas/11 \
  -H "Authorization: Bearer <token_entrenador>"
```

5.4 PUT /api/semanas/{semanaId}

Ruta: /api/semanas/{semanaId}

Método HTTP: PUT

Descripción: Actualiza el número de semana de la entidad Semana con el ID proporcionado.

Seguridad / Acceso:

- Requiere token JWT en encabezado Authorization: Bearer <token>.
- El token debe corresponder a un entrenador.

Headers:

- Authorization: Bearer <token>
- Content-Type: application/json

Path Parameters:

- semanaId (Long, obligatorio): ID de la semana a actualizar.

Request Body (JSON):

```
{
  "numeroSemana": 4
}
```

numeroSemana (int, obligatorio): Nuevo valor para el número de semana.

Flujo interno:

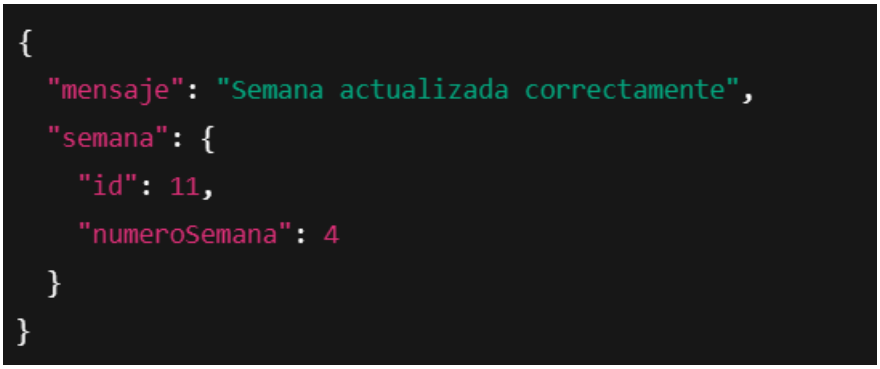
1. **Leer y validar token**:
    - Si falta o inválido → 401 Unauthorized.
2. **Obtener y verificar entrenador**:
    - String correo = jwtUtil.extractEmail(token).
    - entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden.
    - Obtener entrenadorId.
3. **Llamar al servicio**:

- SemanaDTO actualizada = semanaService.actualizarSemana(semanaId, dto).
- En SemanaServiceImpl.actualizarSemana:
    1. Semana semana = semanaRepository.findById(semanaId). Si no existe, lanza IllegalArgumentException("Semana no encontrada").
    2. semana.setNumeroSemana(dto.getNumeroSemana()).
    3. Semana guardada = semanaRepository.save(semana).

4. Retorna new SemanaDTO(guardada.getId(), guardada.getNumeroSemana()).

4. **Devolver respuesta**:

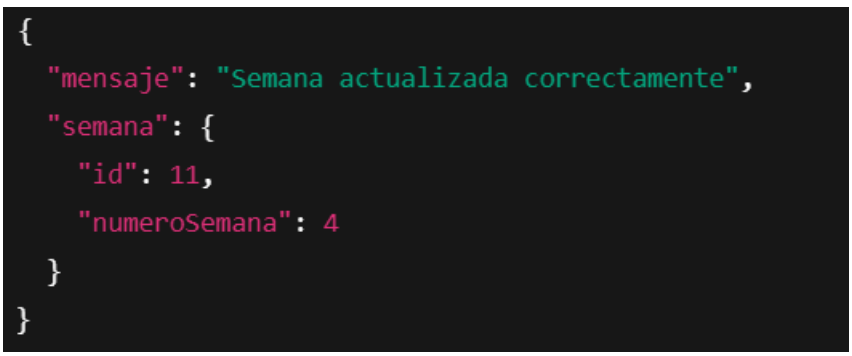
- Si OK: 200 OK con cuerpo JSON:



  - Si IllegalArgumentException: capturado y devuelve 404 Not Found con mensaje “Semana no encontrada” o “Acceso denegado...”.
  - Otros errores → 500 Internal Server Error.

Respuestas posibles:

- **200 OK**:



400 Bad Request / 404 Not Found:

  - “Semana no encontrada”
  - “Acceso denegado: semana no pertenece a este entrenador.” (si se implementa validación extra)

    - “Ya existe una semana con ese número en este bloque.” (si se añade validación de unicidad)
- **401 Unauthorized**: Token ausente o inválido.
- **403 Forbidden**: Token válido pero no corresponde a entrenador.
- **500 Internal Server Error**: Errores imprevistos.

Ejemplo con curl:

```
curl -X PUT http://localhost:8080/api/semanas/11 \
  -H "Authorization: Bearer <token_entrenador>" \
  -H "Content-Type: application/json" \
  -d '{"numeroSemana": 4}'
```

5.5 DELETE /api/semanas/{semanaId}

Ruta: /api/semanas/{semanaId}

Método HTTP: DELETE

Descripción: Elimina la semana con ID semanaId. Esto eliminará, por cascada, sus sub-bloques (y posiblemente los contenidos relacionados).

Seguridad / Acceso:

- Requiere token JWT en encabezado Authorization: Bearer <token>.
- El token debe corresponder a un entrenador.

Headers:

- Authorization: Bearer <token>

Path Parameters:

- semanaId (Long, obligatorio): ID de la semana a eliminar.

Request Body: Ninguno.

Flujo interno:

1. **Leer y validar token**:
    - Si falta o inválido → 401 Unauthorized.

2. **Obtener y verificar entrenador**:
    - String correo = jwtUtil.extractEmail(token).
    - entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden.
    - Obtener entrenadorId.
3. **Llamar al servicio**:

- semanaService.eliminarSemana(semanaId).
- En SemanaServiceImpl.eliminarSemana:
    1. Semana semana = semanaRepository.findById(semanaId). Si no existe, lanza IllegalArgumentException("Semana no encontrada").
    2. semanaRepository.delete(semana). Debido a JPA cascade, se eliminan sub-bloques relacionados.

4. Devolver respuesta:

- Si OK: 200 OK con cuerpo:

```
"Semana eliminada correctamente."
```

- **404 Not Found**:
    - “Semana no encontrada”
    - “Acceso denegado: semana no pertenece a este entrenador.” (si se implementa validación extra)
- **401 Unauthorized**: Token ausente o inválido.
- **403 Forbidden**: Token válido pero no corresponde a entrenador.
- **500 Internal Server Error**: Errores imprevistos.

Ejemplo con curl:

```
curl -X DELETE http://localhost:8080/api/semanas/11 \
  -H "Authorization: Bearer <token_entrenador>"
```

6. Gestión de Sub-bloques (Entrenador)

6.1 POST /api/semanas/{semanaId}/subbloques

Ruta: /api/semanas/{semanaId}/subbloques

Método HTTP: POST

Descripción: Añade un nuevo sub-bloque a la semana identificada por semanaId.

Seguridad / Acceso:

- Requiere token JWT en encabezado Authorization: Bearer <token>.
- El token debe corresponder a un entrenador inscrito en la base de datos.

Headers:

- Authorization: Bearer <token>
- Content-Type: application/json

Path Parameters:

- semanaId (Long, obligatorio): ID de la semana a la que se añade el sub-bloque.

Request Body (JSON):

```
{
  "nombre": "Sub-bloque A"
}
```

nombre (String, obligatorio): Nombre descriptivo del sub-bloque.

Flujo interno:

1. **Leer y validar encabezado Authorization**:
    - Si no existe o no comienza con “Bearer ” → 401 Unauthorized con mensaje “Token no proporcionado.”.
2. **Validar token**:
    - jwtUtil.validateToken(token). Si inválido o expirado → 401 Unauthorized con mensaje “Token inválido o expirado.”.
3. **Obtener correo y verificar entrenador**:
    - String correo = jwtUtil.extractEmail(token).
    - entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden con mensaje “Acceso denegado. Solo entrenadores.”.
4. **Llamar al servicio**:

- SubBloque subBloque = subBloqueService.crearSubBloque(semanaId, dto).
- En SubBloqueServiceImpl.crearSubBloque:
    1. Busca Semana semana = semanaRepository.findById(semanaId). Si no existe, lanza IllegalArgumentException("Semana no encontrada").
    2. Crea SubBloque subBloque = new SubBloque(), asigna nombre y semana.
    3. Guarda: subBloqueRepository.save(subBloque).

5. Devolver respuesta:

- Si OK: 201 Created con cuerpo JSON:

```
{
  "mensaje": "Subbloque añadido correctamente",
  "subBloqueId": 78
}
```

- Si IllegalArgumentException (semana no existe o validación extra de pertenencia falla), se captura en controlador y se devuelve 400 Bad Request con el mensaje de la excepción.
- Otros errores → 500 Internal Server Error.

Respuestas posibles:

- **201 Created**:

```
{
  "mensaje": "Subbloque añadido correctamente",
  "subBloqueId": 78
}
```

400 Bad Request:

- "Semana no encontrada"
- "Acceso denegado: la semana no pertenece a este entrenador." (si se implementa validación extra)
- "Nombre de sub-bloque inválido" u otras validaciones de negocio (p. ej., nombre no vacío).

- **401 Unauthorized**: Token ausente o inválido.
- **403 Forbidden**: Token válido pero no corresponde a un entrenador en BD.
- **500 Internal Server Error**: Errores imprevistos.

Ejemplo con curl:

```
curl -X POST http://localhost:8080/api/semanas/11/subbloques \
  -H "Authorization: Bearer <token_entrenador>" \
  -H "Content-Type: application/json" \
  -d '{"nombre": "Sub-bloque A"}'
```

6.2 GET /api/semanas/{semanaId}/subbloques

Ruta: /api/semanas/{semanaId}/subbloques

Método HTTP: GET

Descripción: Recupera la lista de sub-bloques asociados a la semana con ID semanaId. Devuelve un conjunto de SubBloqueDTO (id y nombre).

Seguridad / Acceso:

- Requiere token JWT en encabezado Authorization: Bearer <token>.
- Token de entrenador válido.

Headers:

- Authorization: Bearer <token>

Path Parameters:

- semanaId (Long, obligatorio): ID de la semana cujos sub-bloques se quieren listar.

Request Body: Ninguno.

Flujo interno:

1. **Validar token**:
    - Si falta o inválido → 401 Unauthorized.
2. **Verificar entrenador**:
    - String correo = jwtUtil.extractEmail(token).
    - entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden.
3. **(Validación recomendada: pertenencia de la semana)**:

- Semana semana = semanaRepository.findById(semanaId). Si no existe → IllegalArgumentException("Semana no encontrada").
- Bloque bloque = semana.getBloque(), comprobar bloque.getEntrenador().getId().equals(entrenador.getId()). Si no → IllegalArgumentException("Acceso denegado: semana no pertenece a este entrenador.").

4. **Llamar al servicio**:
   - Set<SubBloqueDTO> subBloques = subBloqueService.obtenerSubBloquesPorSemana(semanaId).
   - En SubBloqueServiceImpl.obtenerSubBloquesPorSemana:
     1. Semana semana = semanaRepository.findById(semanaId). Si no existe, lanza IllegalArgumentException("Semana no encontrada").
     2. Mapea semana.getSubBloques() a SubBloqueDTO(id, nombre).
5. **Devolver respuesta**:
   - 200 OK con array de objetos:

```
[
  { "id": 78, "nombre": "Sub-bloque A" },
  { "id": 79, "nombre": "Sub-bloque B" }
]
```

   - Si IllegalArgumentException: controlador devuelve 404 Not Found o 400 Bad Request con mensaje apropiado.
   - Otros errores → 500 Internal Server Error.

Respuestas posibles:

- **200 OK**: Lista de sub-bloques (vacía si no hay).

- **400 Bad Request / 404 Not Found**:
    - “Semana no encontrada”
    - “Acceso denegado: semana no pertenece a este entrenador.” (si validación extra)
- **401 Unauthorized**: Token ausente o inválido.
- **403 Forbidden**: Token válido pero no corresponde a entrenador.

Ejemplo con curl:

```
curl -X GET http://localhost:8080/api/semanas/11/subbloques \
  -H "Authorization: Bearer <token_entrenador>"
```

6.3 GET /api/subbloques/{subBloqueId}

Ruta: /api/subbloques/{subBloqueId}

Método HTTP: GET

Descripción: Obtiene los datos de un sub-bloque específico por su ID. Devuelve un SubBloqueDTO con id y nombre.

Seguridad / Acceso:

- Requiere token JWT en encabezado Authorization: Bearer <token>.
- Token de entrenador válido.

Headers:

- Authorization: Bearer <token>

Path Parameters:

- subBloqueId (Long, obligatorio): ID del sub-bloque a recuperar.

Request Body: Ninguno.

Flujo interno:

1. **Validar token**:
    - Si falta o inválido → 401 Unauthorized.

2. **Verificar entrenador**:
    - String correo = jwtUtil.extractEmail(token).
    - entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden.
3. **Llamar al servicio**:
    - SubBloqueDTO subBloque = subBloqueService.obtenerSubBloque(subBloqueId).
    - En SubBloqueServiceImpl.obtenerSubBloque:
        1. SubBloque sb = subBloqueRepository.findById(subBloqueId). Si no existe, lanza IllegalArgumentException("Subbloque no encontrado").
        2. Retorna new SubBloqueDTO(sb.getId(), sb.getNombre()).
4. **(Validación recomendada: pertenencia)**:
    - Recuperar entidad completa SubBloque sb, luego Semana semana = sb.getSemana(), Bloque bloque = semana.getBloque(), y verificar bloque.getEntrenador().getId().equals(entrenador.getId()). Si no, lanzar IllegalArgumentException("Acceso denegado: sub-bloque no pertenece a este entrenador.").
5. **Devolver respuesta**:
    - 200 OK con cuerpo:


```
{
  "id": 78,
  "nombre": "Sub-bloque A"
}
```


    - Si IllegalArgumentException: 404 Not Found o 400 Bad Request con mensaje “Subbloque no encontrado” o “Acceso denegado...”.

- Otros errores → 500 Internal Server Error.

Respuestas posibles:

- **200 OK**:

```
{
  "id": 78,
  "nombre": "Sub-bloque A"
}
```

404 Not Found / 400 Bad Request:

- "Subbloque no encontrado"
- "Acceso denegado: sub-bloque no pertenece a este entrenador." (si se implementa validación extra)

- **401 Unauthorized**: Token ausente o inválido.
- **403 Forbidden**: Token válido pero no corresponde a entrenador.

Ejemplo con curl:

```
curl -X GET http://localhost:8080/api/subbloques/78 \
  -H "Authorization: Bearer <token_entrenador>"
```

6.4 PUT /api/subbloques/{subBloqueId}

Ruta: /api/subbloques/{subBloqueId}

Método HTTP: PUT

Descripción: Actualiza el nombre de un sub-bloque existente.

Seguridad / Acceso:

- Requiere token JWT en encabezado Authorization: Bearer <token>.
- Token de entrenador válido.

Headers:

- Authorization: Bearer <token>

- Content-Type: application/json

Path Parameters:

- subBloqueId (Long, obligatorio): ID del sub-bloque a actualizar.

Request Body (JSON):

```
{
  "nombre": "Nuevo nombre de sub-bloque"
}
```

nombre (String, obligatorio): Nuevo nombre.

Flujo interno:

1. **Validar token**:
    - Si falta o inválido → 401 Unauthorized.
2. **Verificar entrenador**:
    - String correo = jwtUtil.extractEmail(token).
    - entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden.
3. **(Validación extra: pertenencia)**:
    - SubBloque sb = subBloqueRepository.findById(subBloqueId). Si no existe → IllegalArgumentException("Subbloque no encontrado").
    - Verificar sb.getSemana().getBloque().getEntrenador().getId().equals(entrenador.getId()). Si no → IllegalArgumentException("Acceso denegado: sub-bloque no pertenece a este entrenador.").
4. **Llamar al servicio**:
    - SubBloqueDTO actualizado = subBloqueService.actualizarSubBloque(subBloqueId, dto).
    - En SubBloqueServiceImpl.actualizarSubBloque:

1. Busca SubBloque subBloque = subBloqueRepository.findById(subBloqueId). Si no existe, lanza IllegalArgumentException("Subbloque no encontrado").
2. subBloque.setNombre(dto.getNombre()).
3. Guarda: SubBloque guardado = subBloqueRepository.save(subBloque).
4. Retorna new SubBloqueDTO(guardado.getId(), guardado.getNombre()).

5. **Devolver respuesta**:
   - Si OK: 200 OK con cuerpo JSON:

```
{
  "mensaje": "Subbloque actualizado correctamente",
  "subBloque": {
    "id": 78,
    "nombre": "Nuevo nombre de sub-bloque"
  }
}
```

   - Si IllegalArgumentException: 400 Bad Request o 404 Not Found con mensaje apropiado.
   - Otros errores → 500 Internal Server Error.

Respuestas posibles:

- **200 OK**:

```
{
  "mensaje": "Subbloque actualizado correctamente",
  "subBloque": {
    "id": 78,
    "nombre": "Nuevo nombre de sub-bloque"
  }
}
```

400 Bad Request / 404 Not Found:

- “Subbloque no encontrado”
- “Acceso denegado: sub-bloque no pertenece a este entrenador.” (si validación extra)
- “Ya existe un sub-bloque con ese nombre en esta semana.” (si se implementa validación de unicidad)

- **401 Unauthorized**: Token ausente o inválido.
- **403 Forbidden**: Token válido pero no corresponde a entrenador.

Ejemplo con curl:

```
curl -X PUT http://localhost:8080/api/subbloques/78 \
  -H "Authorization: Bearer <token_entrenador>" \
  -H "Content-Type: application/json" \
  -d '{"nombre": "Nuevo nombre de sub-bloque"}'
```

6.5 DELETE /api/subbloques/{subBloqueId}

Ruta: /api/subbloques/{subBloqueId}

Método HTTP: DELETE

Descripción: Elimina el sub-bloque con ID subBloqueId. Por cascada, se eliminarán entidades hijas (por ejemplo, ejercicios sugeridos, series de entrenamiento asociadas, etc.), según configuración de JPA.

Seguridad / Acceso:

- Requiere token JWT en encabezado Authorization: Bearer <token>.
- Token de entrenador válido.

Headers:

- Authorization: Bearer <token>

Path Parameters:

- subBloqueId (Long, obligatorio): ID del sub-bloque a eliminar.

Request Body: Ninguno.

Flujo interno:

1. **Validar token**:
    - Si falta o inválido → 401 Unauthorized.
2. **Verificar entrenador**:
    - String correo = jwtUtil.extractEmail(token).
    - entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden.
3. **(Validación extra: pertenencia)**:
    - SubBloque sb = subBloqueRepository.findById(subBloqueId). Si no existe → IllegalArgumentException("Subbloque no encontrado").
    - Verificar sb.getSemana().getBloque().getEntrenador().getId().equals(entrenador.getId()). Si no → IllegalArgumentException("Acceso denegado: sub-bloque no pertenece a este entrenador.").
4. **Llamar al servicio**:
    - subBloqueService.eliminarSubBloque(subBloqueId).
    - En SubBloqueServiceImpl.eliminarSubBloque:
        1. SubBloque subBloque = subBloqueRepository.findById(subBloqueId). Si no existe, lanza IllegalArgumentException("Subbloque no encontrado").
        2. subBloqueRepository.delete(subBloque).
5. **Devolver respuesta**:
    - Si OK: 200 OK con cuerpo:

```
"Subbloque eliminado correctamente."
```

1.

   - Si IllegalArgumentException: 404 Not Found o 400 Bad Request con mensaje “Subbloque no encontrado” o “Acceso denegado...”.
   - Otros errores → 500 Internal Server Error.

Respuestas posibles:

- **200 OK**:

```
"Subbloque eliminado correctamente."
```

404 Not Found / 400 Bad Request:

   - “Subbloque no encontrado”
   - “Acceso denegado: sub-bloque no pertenece a este entrenador.” (si validación extra)

- **401 Unauthorized**: Token ausente o inválido.
- **403 Forbidden**: Token válido, pero no corresponde a entrenador.

Ejemplo con curl:

```
curl -X DELETE http://localhost:8080/api/subbloques/78 \
  -H "Authorization: Bearer <token_entrenador>"
```

7. Gestión de Ejercicios Sugeridos (Entrenador)

7.1 POST /api/subbloques/{subBloqueId}/ejercicios-sugeridos

Ruta: /api/subbloques/{subBloqueId}/ejercicios-sugeridos

Método HTTP: POST

Descripción: Añade un nuevo ejercicio sugerido al sub-bloque identificado por subBloqueId, junto con sus series sugeridas (opcional).

🔒 Seguridad / Acceso:

- **Requiere token JWT en encabezado Authorization: Bearer <token>.**
- **Token de entrenador válido.**

Headers:

- **Authorization: Bearer <token>**
- **Content-Type: application/json**

Path Parameters:

- **subBloqueId (Long, obligatorio): ID del sub-bloque donde se crea el ejercicio sugerido.**

Request Body (JSON):

```
{
  "nombre": "Press de banca",
  "grupoMuscular": "Pecho",
  "series": [
    { "numeroSerie": 1, "repeticiones": 10, "peso": 60.0 },
    { "numeroSerie": 2, "repeticiones": 8, "peso": 65.0 }
  ]
}
```

- **nombre (String, obligatorio): Nombre del ejercicio.**
- **grupoMuscular (String, opcional o obligatorio según negocio): Grupo muscular objetivo.**
- **series (Set<SerieSugeridaDTO>, opcional): Conjunto de series sugeridas iniciales. Cada SerieSugeridaDTO contiene:**
    - **numeroSerie (int)**
    - **repeticiones (Integer)**
    - **peso (Double)**

Flujo interno:

1. **Validar token:**
    - **Si falta o inválido → 401 Unauthorized.**
2. **Verificar entrenador:**
    - **String correo = jwtUtil.extractEmail(token).**
    - **entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden.**
3. **(Validación recomendada: pertenencia del sub-bloque):**
    - **SubBloque sb = subBloqueRepository.findById(subBloqueId). Si no existe → IllegalArgumentException("Subbloque no encontrado").**
    - **Semana semana = sb.getSemana(), Bloque bloque = semana.getBloque(), verificar bloque.getEntrenador().getId().equals(entrenador.getId()). Si no → IllegalArgumentException("Acceso denegado: sub-bloque no pertenece a este entrenador.").**
4. **Llamar al servicio:**
    - **EjercicioSugerido ejercicio = ejercicioSugeridoService.agregarEjercicioSugerido(subBloqueId, dto).**
    - **En EjercicioSugeridoServiceImpl.agregarEjercicioSugerido:**
        1. **Busca SubBloque subBloque = subBloqueRepository.findById(subBloqueId). Si no existe, lanza IllegalArgumentException("Subbloque no encontrado").**
        2. **Crea EjercicioSugerido ejercicio = new EjercicioSugerido(), asigna nombre, grupoMuscular, subBloque.**

3. **Guarda: ejercicioSugeridoRepository.save(ejercicio).**
4. **Series iniciales: Actualmente, el servicio no procesa el campo series del DTO en la creación. Si se desea soportarlo, habría que:**
    - **Para cada SerieSugeridaDTO dtoSerie en dto.getSeries(), crear SerieSugerida con numeroSerie, repeticiones, peso, asociarlo a ejercicio, y guardar en cascada (si la relación en EjercicioSugerido está configurada con cascade). En la implementación actual, no se maneja; se guarda el ejercicio sin series. Se podría extender el servicio para manejar series iniciales si el DTO las incluye.**

5. **Devolver respuesta:**
    - **201 Created con cuerpo JSON:**

```
{
  "mensaje": "Ejercicio sugerido añadido correctamente",
  "ejercicioSugeridoId": 55
}
```

    - **Si IllegalArgumentException: 400 Bad Request con mensaje "Subbloque no encontrado" o "Acceso denegado..." o validaciones: nombre obligatorio.**
    - **Otros errores → 500 Internal Server Error.**

□ Respuestas posibles:

- **201 Created:**

```
{
  "mensaje": "Ejercicio sugerido añadido correctamente",
  "ejercicioSugeridoId": 55
}
```

400 Bad Request:

- **"Subbloque no encontrado"**
- **"Acceso denegado: sub-bloque no pertenece a este entrenador." (si validación extra)**
- **"Nombre de ejercicio sugerido obligatorio" (si se agrega validación).**

401 Unauthorized, 403 Forbidden, 500 Internal Server Error según los casos.

```
curl -X POST http://localhost:8080/api/subbloques/78/ejercicios-sugeridos \
  -H "Authorization: Bearer <token_entrenador>" \
  -H "Content-Type: application/json" \
  -d '{
    "nombre": "Press de banca",
    "grupoMuscular": "Pecho",
    "series": [
      { "numeroSerie": 1, "repeticiones": 10, "peso": 60.0 },
      { "numeroSerie": 2, "repeticiones": 8, "peso": 65.0 }
    ]
  }'
```

7.2 GET /api/subbloques/{subBloqueId}/ejercicios-sugeridos

Ruta: /api/subbloques/{subBloqueId}/ejercicios-sugeridos

Método HTTP: GET

Descripción: Lista los ejercicios sugeridos que existen en el sub-bloque identificado por subBloqueId. Devuelve un conjunto de EjercicioSugeridoDTO (id, nombre, grupoMuscular).

Seguridad / Acceso:

- **Requiere token JWT en encabezado Authorization: Bearer <token>.**
- **Token de entrenador válido.**

Headers:

- **Authorization: Bearer <token>**

Path Parameters:

- **subBloqueId (Long, obligatorio): ID del sub-bloque cuyos ejercicios sugeridos se listan.**

Request Body: Ninguno.

Flujo interno:

1. **Validar token:**

- **Si falta o inválido → 401 Unauthorized.**

2. **Verificar entrenador:**
   - **String correo = jwtUtil.extractEmail(token).**
   - **entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden.**
3. **(Validación recomendada: pertenencia del sub-bloque):**
   - **SubBloque sb = subBloqueRepository.findById(subBloqueId). Si no existe → IllegalArgumentException("Subbloque no encontrado").**
   - **Verificar sb.getSemana().getBloque().getEntrenador().getId().equals(entrenador.getId()). Si no → IllegalArgumentException("Acceso denegado: sub-bloque no pertenece a este entrenador.").**
4. **Llamar al servicio:**
   - **Set<EjercicioSugeridoDTO> ejercicios = ejercicioSugeridoService.obtenerEjerciciosPorSubBloque(subBloqueId).**
   - **En EjercicioSugeridoServiceImpl.obtenerEjerciciosPorSubBloque:**
     1. **Busca SubBloque subBloque = subBloqueRepository.findById(subBloqueId). Si no existe, lanza IllegalArgumentException("Subbloque no encontrado").**
     2. **Mapea subBloque.getEjerciciosSugeridos() a EjercicioSugeridoDTO(id, nombre, grupoMuscular).**
5. **Devolver respuesta:**
   - **200 OK con cuerpo JSON:**

```
[
  { "id": 55, "nombre": "Press de banca", "grupoMuscular": "Pecho" },
  { "id": 56, "nombre": "Fondos", "grupoMuscular": "Tríceps" }
]
```

   - **Si IllegalArgumentException: 404 Not Found o 400 Bad Request con mensaje.**
   - **Otros errores → 500 Internal Server Error.**

Respuestas posibles:

- **200 OK: Lista de ejercicios sugeridos (vacía si no hay).**
- **400 Bad Request / 404 Not Found:**
    - **“Subbloque no encontrado”**
    - **“Acceso denegado: sub-bloque no pertenece a este entrenador.” (si validación extra)**
- **401 Unauthorized, 403 Forbidden, 500 Internal Server Error.**

Ejemplo con curl:

```
curl -X GET http://localhost:8080/api/subbloques/78/ejercicios-sugeridos \
  -H "Authorization: Bearer <token_entrenador>"
```

7.3 GET /api/subbloques/ejercicios-sugeridos/{id}

Ruta: /api/subbloques/ejercicios-sugeridos/{id}

Método HTTP: GET

Descripción: Obtiene los datos de un ejercicio sugerido por su ID. Devuelve EjercicioSugeridoDTO con id, nombre y grupoMuscular.

Seguridad / Acceso:

- **Requiere token JWT en encabezado Authorization: Bearer <token>.**
- **Token de entrenador válido.**

Headers:

- **Authorization: Bearer <token>**

Path Parameters:

- **id (Long, obligatorio): ID del ejercicio sugerido a recuperar.**

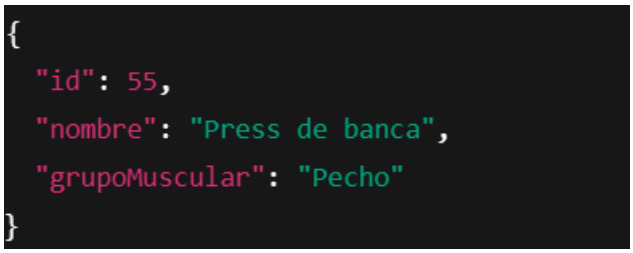
Request Body: Ninguno.

Flujo interno:

1. **Validar token:**
    - **Si falta o inválido → 401 Unauthorized.**
2. **Verificar entrenador:**
    - **String correo = jwtUtil.extractEmail(token).**

- entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden.

3. Llamar al servicio:
   - EjercicioSugeridoDTO dto = ejercicioSugeridoService.obtenerEjercicioSugerido(id).
   - En EjercicioSugeridoServiceImpl.obtenerEjercicioSugerido:
     1. EjercicioSugerido ejercicio = ejercicioSugeridoRepository.findById(id). Si no existe, lanza IllegalArgumentException("Ejercicio sugerido no encontrado").
     2. Mapea a DTO.
4. (Validación recomendada: pertenencia):
   - Recuperar EjercicioSugerido ejercicio, luego SubBloque sb = ejercicio.getSubBloque(), Semana semana = sb.getSemana(), Bloque bloque = semana.getBloque(), y verificar bloque.getEntrenador().getId().equals(entrenador.getId()). Si no → IllegalArgumentException("Acceso denegado: ejercicio sugerido no pertenece a este entrenador.").
5. Devolver respuesta:
   - 200 OK con cuerpo:



   - Si IllegalArgumentException: 404 Not Found con mensaje “Ejercicio sugerido no encontrado” o “Acceso denegado...”.
   - Otros errores → 500 Internal Server Error.

Respuestas posibles:

- 200 OK:

```
{
  "id": 55,
  "nombre": "Press de banca",
  "grupoMuscular": "Pecho"
}
```

- **404 Not Found / 400 Bad Request:**
    - **“Ejercicio sugerido no encontrado”**
    - **“Acceso denegado: ejercicio sugerido no pertenece a este entrenador.” (si validación extra)**
- **401 Unauthorized, 403 Forbidden, 500 Internal Server Error.**

Ejemplo con curl:

```
curl -X GET http://localhost:8080/api/subbloques/ejercicios-sugeridos/55 \
  -H "Authorization: Bearer <token_entrenador>"
```

7.4 PUT /api/subbloques/ejercicios-sugeridos/{id}

Ruta: /api/subbloques/ejercicios-sugeridos/{id}

Método HTTP: PUT

Descripción: Actualiza los datos de un ejercicio sugerido existente (nombre y grupo muscular). No actualiza series; para series existen endpoints separados.

Seguridad / Acceso:

- **Requiere token JWT en encabezado Authorization: Bearer <token>.**
- **Token de entrenador válido.**

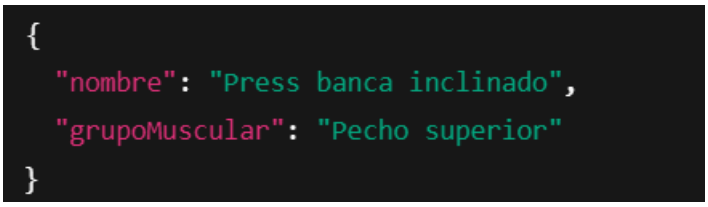
Headers:

- **Authorization: Bearer <token>**
- **Content-Type: application/json**

Path Parameters:

- **id (Long, obligatorio): ID del ejercicio sugerido a actualizar.**

Request Body (JSON):



- **nombre (String, obligatorio): Nuevo nombre.**
- **grupoMuscular (String, opcional/obligatorio según política).**

Flujo interno:

1. **Validar token:**
    - **Si falta o inválido → 401 Unauthorized.**
2. **Verificar entrenador:**
    - **String correo = jwtUtil.extractEmail(token).**
    - **entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden.**
3. **(Validación de pertenencia):**
    - **EjercicioSugerido ejercicio = ejercicioSugeridoRepository.findById(id). Si no existe → IllegalArgumentException("Ejercicio sugerido no encontrado").**
    - **Verificar ejercicio.getSubBloque().getSemana().getBloque().getEntrenador().getId().equals(entrenador.getId()). Si no → IllegalArgumentException("Acceso denegado: ejercicio sugerido no pertenece a este entrenador.").**
4. **Llamar al servicio:**
    - **EjercicioSugeridoDTO actualizado = ejercicioSugeridoService.actualizarEjercicioSugerido(id, dto).**
    - **En EjercicioSugeridoServiceImpl.actualizarEjercicioSugerido:**
        1. **Busca EjercicioSugerido ejercicio = ejercicioSugeridoRepository.findById(id). Si no existe, lanza IllegalArgumentException("Ejercicio sugerido no encontrado").**
        2. **ejercicio.setNombre(dto.getNombre()), ejercicio.setGrupoMuscular(dto.getGrupoMuscular()).**

3. **Guarda: EjercicioSugerido guardado = ejercicioSugeridoRepository.save(ejercicio).**
4. **Retorna EjercicioSugeridoDTO(mapToDTO).**

5. **Devolver respuesta:**
   - **200 OK con cuerpo:**

```
{
  "id": 55,
  "nombre": "Press banca inclinado",
  "grupoMuscular": "Pecho superior"
}
```

   - **El controlador actual envuelve respuesta en ResponseEntity.ok(...) directamente con DTO, o en caso de error 400 Bad Request con mensaje.**

Respuestas posibles:

- **200 OK:**

```
{
  "id": 55,
  "nombre": "Press banca inclinado",
  "grupoMuscular": "Pecho superior"
}
```

- **400 Bad Request / 404 Not Found:**
  - **“Ejercicio sugerido no encontrado”**
  - **“Acceso denegado: ejercicio sugerido no pertenece a este entrenador.” (si validación extra)**
  - **“Nombre obligatorio” u otras validaciones.**
- **401 Unauthorized, 403 Forbidden, 500 Internal Server Error.**

Ejemplo con curl:

```
curl -X PUT http://localhost:8080/api/subbloques/ejercicios-sugeridos/55 \
  -H "Authorization: Bearer <token_entrenador>" \
  -H "Content-Type: application/json" \
  -d '{"nombre": "Press banca inclinado","grupoMuscular": "Pecho superior"}'
```

7.5 DELETE /api/subbloques/ejercicios-sugeridos/{id}

Ruta: /api/subbloques/ejercicios-sugeridos/{id}

Método HTTP: DELETE

Descripción: Elimina el ejercicio sugerido con ID id de la base de datos. Al eliminarlo, se eliminarán en cascada las series sugeridas asociadas a ese ejercicio (según la configuración JPA).

Seguridad / Acceso:

- **Requiere token JWT en encabezado Authorization: Bearer <token>.**
- **Token de entrenador válido.**

Headers:

- **Authorization: Bearer <token>**

Path Parameters:

- **id (Long, obligatorio): ID del ejercicio sugerido a eliminar.**

Request Body: Ninguno.

Flujo interno:

1. **Validar token:**
    - **Si falta o inválido → 401 Unauthorized.**
2. **Verificar entrenador:**
    - **String correo = jwtUtil.extractEmail(token).**
    - **entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden.**
3. **(Validación de pertenencia):**
    - **EjercicioSugerido ejercicio = ejercicioSugeridoRepository.findById(id). Si no existe → IllegalArgumentException("Ejercicio sugerido no encontrado").**
    - **Verificar ejercicio.getSubBloque().getSemana().getBloque().getEntrenador().getId().equals(entrenador.getId()). Si no →**

IllegalArgumentException("Acceso denegado: ejercicio sugerido no pertenece a este entrenador.").

4. **Llamar al servicio:**
    - **ejercicioSugeridoService.eliminarEjercicioSugerido(id).**
    - **En EjercicioSugeridoServiceImpl.eliminarEjercicioSugerido:**
        1. **Busca EjercicioSugerido ejercicio = ejercicioSugeridoRepository.findById(id). Si no existe, lanza IllegalArgumentException("Ejercicio sugerido no encontrado").**
        2. **ejercicioSugeridoRepository.delete(ejercicio).**
5. **Devolver respuesta:**
    - **200 OK con cuerpo:**

```
"Ejercicio eliminado correctamente."
```

1.
    - **Si IllegalArgumentException: 404 Not Found o 400 Bad Request con mensaje.**
    - **Otros errores → 500 Internal Server Error.**

Respuestas posibles:

- **200 OK:**

```
"Ejercicio eliminado correctamente."
```

- **404 Not Found / 400 Bad Request:**
    - **“Ejercicio sugerido no encontrado”**
    - **“Acceso denegado: ejercicio sugerido no pertenece a este entrenador.” (si validación extra)**
- **401 Unauthorized, 403 Forbidden, 500 Internal Server Error.**

Ejemplo con curl:

```
curl -X DELETE http://localhost:8080/api/subbloques/ejercicios-sugeridos/55 \
  -H "Authorization: Bearer <token_entrenador>"
```

8. Gestión de Series Sugeridas (Entrenador)

8.1 GET /api/series-sugeridas/ejercicios-sugeridos/{ejercicioId}/series

Ruta: /api/series-sugeridas/ejercicios-sugeridos/{ejercicioId}/series

Método HTTP: GET

Descripción: Recupera todas las series sugeridas asociadas al ejercicio sugerido identificado por ejercicioId.

Seguridad / Acceso:

- **Requiere token JWT en encabezado Authorization: Bearer <token>.**
- **El token debe corresponder a un entrenador válido.**

Headers:

- **Authorization: Bearer <token>**

Path Parameters:

- **ejercicioId (Long, obligatorio): ID del ejercicio sugerido cuyas series se listan.**

Request Body: Ninguno.

Flujo interno:

1. **Validar token:**
    - **Si falta o no inicia con Bearer → 401 Unauthorized ("Token no proporcionado o inválido.").**
    - **Extraer substring del token y jwtUtil.validateToken(token). Si inválido o expirado → 401 Unauthorized ("Token inválido o expirado.").**
2. **Verificar entrenador:**
    - **String correo = jwtUtil.extractEmail(token).**
    - **entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden ("Acceso denegado. Solo entrenadores.").**
3. **(Validación de pertenencia):**

- **EjercicioSugerido ejercicio = ejercicioSugeridoRepository.findById(ejercicioId). Si no existe → 404 Not Found con mensaje "Ejercicio sugerido no encontrado".**
- **Verificar que el ejercicio pertenece a un sub-bloque de un bloque de este entrenador:**

```
SubBloque sb = ejercicio.getSubBloque();
Bloque bloque = sb.getSemana().getBloque();
if (!bloque.getEntrenador().getId().equals(entrenador.getId())) {
    throw new IllegalArgumentException("Acceso denegado: ejercicio sugerido no perten
}
```

Si falla la verificación, devolver 403 Forbidden o 400 Bad Request con mensaje apropiado.

4 Llamar al servicio:

- **Set<SerieSugeridaDTO> series = serieSugeridaService.obtenerSeriesPorEjercicio(ejercicioId).**
- **En SerieSugeridaServiceImpl.obtenerSeriesPorEjercicio:**
  1. **Busca EjercicioSugerido ejercicio = ejercicioSugeridoRepository.findById(ejercicioId). Si no existe, lanza IllegalArgumentException("Ejercicio sugerido no encontrado").**
  2. **Recupera ejercicio.getSeriesSugeridas(), mapea cada SerieSugerida a SerieSugeridaDTO(id, numeroSerie, repeticiones, peso).**

     Devolver respuesta:

- **200 OK con cuerpo: lista JSON de objetos SerieSugeridaDTO:**

```
[
  { "id": 101, "numeroSerie": 1, "repeticiones": 10, "peso": 50.0 },
  { "id": 102, "numeroSerie": 2, "repeticiones": 8,  "peso": 55.0 }
]
```

- **Si no hay series, devuelve array vacío [].**
- **Si IllegalArgumentException por ejercicio no encontrado → 404 Not Found con mensaje.**
- **Si validación de pertenencia falla → 403 Forbidden o 400 Bad Request según convención.**
- **Otros errores → 500 Internal Server Error.**

Respuestas posibles:

- **200 OK: Array de series (posiblemente vacío).**
- **401 Unauthorized: Token ausente o inválido.**
- **403 Forbidden: Token válido pero no es entrenador o no le pertenece el recurso.**
- **404 Not Found: Ejercicio sugerido no encontrado.**
- **500 Internal Server Error: Errores imprevistos.**

Ejemplo con curl:

```
curl -X GET http://localhost:8080/api/series-sugeridas/ejercicios-sugeridos/55/series \
  -H "Authorization: Bearer <token_entrenador>"
```

8.2 POST /api/series-sugeridas/ejercicios-sugeridos/{ejercicioId}/series

Ruta: /api/series-sugeridas/ejercicios-sugeridos/{ejercicioId}/series

Método HTTP: POST

Descripción: Añade una nueva serie sugerida a un ejercicio sugerido existente.

Seguridad / Acceso:

- **Requiere token JWT en encabezado Authorization: Bearer <token>.**
- **Token de entrenador válido.**

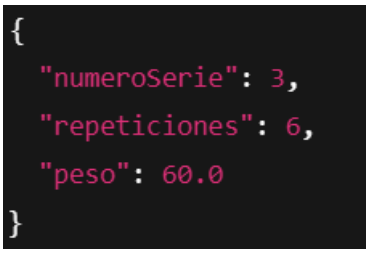
Headers:

- **Authorization: Bearer <token>**
- **Content-Type: application/json**

Path Parameters:

- **ejercicioId (Long, obligatorio): ID del ejercicio sugerido al que se asocia la nueva serie.**

Request Body (JSON):



- **numeroSerie (int, obligatorio): Número de la serie.**
- **repeticiones (Integer, opcional u obligatorio según política): Número de repeticiones recomendadas.**
- **peso (Double, opcional u obligatorio según política): Peso sugerido.**

Flujo interno:

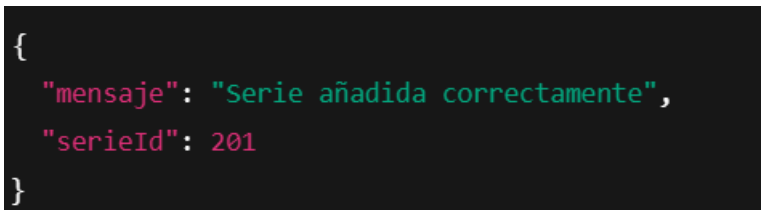
1. **Validar token:**
    - **Si falta o inválido → 401 Unauthorized.**
2. **Verificar entrenador:**
    - **String correo = jwtUtil.extractEmail(token).**
    - **entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden.**
3. **Validación de existencia y pertenencia del ejercicio:**
    - **EjercicioSugerido ejercicio = ejercicioSugeridoRepository.findById(ejercicioId). Si no existe → 404 Not Found con mensaje "Ejercicio sugerido no encontrado".**
    - **Verificar que ejercicio.getSubBloque().getSemana().getBloque().getEntrenador().getId().equals(entrenador.getId()). Si no, devolver 403 Forbidden o 400 Bad Request.**
4. **Llamar al servicio:**
    - **Construir DTO: SerieSugeridaDTO dto con los campos del body.**
    - **SerieSugerida nueva = serieSugeridaService.agregarSerieAEjercicio(ejercicioId, dto).**
    - **En SerieSugeridaServiceImpl.agregarSerieAEjercicio:**
        1. **Busca EjercicioSugerido ejercicio = ejercicioSugeridoRepository.findById(ejercicioId). Si no existe,**

lanza IllegalArgumentException("Ejercicio sugerido no encontrado").

2. **Crea SerieSugerida serie = new SerieSugerida(), asigna numeroSerie, repeticiones, peso, serie.setEjercicioSugerido(ejercicio).**
3. **Guarda: serieSugeridaRepository.save(serie).**

- **Retorna la entidad guardada (con ID).**

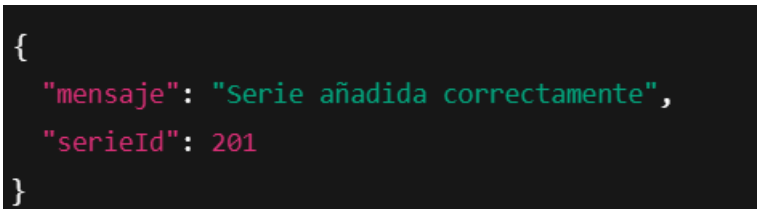
5. **Devolver respuesta:**
   - **201 Created con cuerpo JSON:**



   - **Si IllegalArgumentException: 400 Bad Request con mensaje, o 404 Not Found si ejercicio no existe.**
   - **Si validación de pertenencia falla: 403 Forbidden.**
   - **Otros errores → 500 Internal Server Error.**

Respuestas posibles:

- **201 Created:**



- **400 Bad Request:**
  - **“Ejercicio sugerido no encontrado”**
  - **“Acceso denegado: ejercicio sugerido no pertenece a este entrenador.”**
  - **Validaciones de campos, p.ej. “numeroSerie obligatorio”.**
- **401 Unauthorized, 403 Forbidden, 500 Internal Server Error.**

Ejemplo con curl:

```
curl -X POST http://localhost:8080/api/series-sugeridas/ejercicios-sugeridos/55/series \
  -H "Authorization: Bearer <token_entrenador>" \
  -H "Content-Type: application/json" \
  -d '{"numeroSerie": 3, "repeticiones": 6, "peso": 60.0}'
```

8.3 PUT /api/series-sugeridas/{serieId}

Ruta: /api/series-sugeridas/{serieId}

Método HTTP: PUT

Descripción: Actualiza una serie sugerida existente identificada por serieId. Solo modifica campos numeroSerie, repeticiones y peso.

Seguridad / Acceso:

- **Requiere token JWT en encabezado Authorization: Bearer <token>.**
- **Token de entrenador válido.**

Headers:

- **Authorization: Bearer <token>**
- **Content-Type: application/json**

Path Parameters:

- **serieId (Long, obligatorio): ID de la serie sugerida a actualizar.**

Request Body (JSON):

```
{
  "numeroSerie": 2,
  "repeticiones": 12,
  "peso": 55.0
}
```

- **Campos similares a creación.**

Flujo interno:

1. **Validar token:**
    - **Si falta o inválido → 401 Unauthorized.**
2. **Verificar entrenador:**

- **String correo = jwtUtil.extractEmail(token).**
- **entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden.**

3. **Validación de existencia y pertenencia de la serie:**
   - **SerieSugerida serie = serieSugeridaRepository.findById(serieId). Si no existe → 404 Not Found con mensaje "Serie sugerida no encontrada.".**
   - **Recuperar ejercicio asociado: EjercicioSugerido ejercicio = serie.getEjercicioSugerido(), verificar que ejercicio.getSubBloque().getSemana().getBloque().getEntrenador().getId().equals(entrenador.getId()). Si no → 403 Forbidden.**
4. **Llamar al servicio:**
   - **Construir entidad temporal o DTO con nuevos valores.**
   - **SerieSugerida actualizada = serieSugeridaService.actualizarSerie(serieId, nuevaData).**
   - **En SerieSugeridaServiceImpl.actualizarSerie:**
     1. **Busca SerieSugerida serie = serieSugeridaRepository.findById(id). Si no existe, lanza IllegalArgumentException("Serie sugerida no encontrada.").**
     2. **serie.setNumeroSerie(nuevaData.getNumeroSerie()), serie.setRepeticiones(...), serie.setPeso(...).**
     3. **Guarda: serieSugeridaRepository.save(serie).**
5. **Devolver respuesta:**
   - **200 OK con cuerpo:**

```
"Serie actualizada correctamente."
```

   - **Si IllegalArgumentException: 404 Not Found o 400 Bad Request con mensaje.**
   - **Si pertenencia incorrecta: 403 Forbidden.**
   - **Otros errores → 500 Internal Server Error.**

Respuestas posibles:

- **200 OK:**

```
"Serie actualizada correctamente."
```

- **404 Not Found:**
  - **“Serie sugerida no encontrada.”**
- **400 Bad Request:**
  - **Validaciones de campos.**
- **401 Unauthorized, 403 Forbidden, 500 Internal Server Error.**

Ejemplo con curl:

```
curl -X PUT http://localhost:8080/api/series-sugeridas/201 \
  -H "Authorization: Bearer <token_entrenador>" \
  -H "Content-Type: application/json" \
  -d '{"numeroSerie": 2, "repeticiones": 12, "peso": 55.0}'
```

8.4 DELETE /api/series-sugeridas/{serieId}

Ruta: /api/series-sugeridas/{serieId}

Método HTTP: DELETE

Descripción: Elimina la serie sugerida identificada por serieId.

Seguridad / Acceso:

- **Requiere token JWT en encabezado Authorization: Bearer <token>.**
- **Token de entrenador válido.**

Headers:

- **Authorization: Bearer <token>**

Path Parameters:

- **serieId (Long, obligatorio): ID de la serie sugerida a eliminar.**

Request Body: Ninguno.

Flujo interno:

1. **Validar token: Si falta o inválido → 401 Unauthorized.**

2. **Verificar entrenador: String correo = jwtUtil.extractEmail(token). entrenadorRepository.findByCorreo(correo). Si no existe → 403 Forbidden.**
3. **Validar existencia y pertenencia:**
    - **SerieSugerida serie = serieSugeridaRepository.findById(serieId). Si no existe → 404 Not Found ("Serie sugerida no encontrada.").**
    - **Verificar que la serie pertenece a un ejercicio sugerido de un sub-bloque de un bloque de este entrenador:**

```
EjercicioSugerido ejercicio = serie.getEjercicioSugerido();
if (!ejercicio.getSubBloque().getSemana().getBloque().getEntrenador().getId().equals(
    throw new IllegalArgumentException("Acceso denegado: serie sugerida no pertenece
}
```

- **Si falla, devolver 403 Forbidden.**

Llamar al servicio:

- **serieSugeridaService.eliminarSerie(serieId).**
- **En SerieSugeridaServiceImpl.eliminarSerie:**
    1. **Busca SerieSugerida serie = serieSugeridaRepository.findById(id). Si no existe, lanza IllegalArgumentException("Serie sugerida no encontrada.").**
    2. **serieSugeridaRepository.delete(serie).**

Devolver respuesta:

- **200 OK con cuerpo:**

```
"Serie eliminada correctamente."
```

    - **Si IllegalArgumentException: 404 Not Found o 400 Bad Request con mensaje.**
    - **Si pertenencia incorrecta: 403 Forbidden.**
    - **Otros errores → 500 Internal Server Error.**

Respuestas posibles:

- **200 OK:**

```
"Serie eliminada correctamente."
```

- **404 Not Found:**

- **“Serie sugerida no encontrada.”**

- **401 Unauthorized, 403 Forbidden, 500 Internal Server Error.**

Ejemplo con curl:

```
curl -X DELETE http://localhost:8080/api/series-sugeridas/201 \
  -H "Authorization: Bearer <token_entrenador>"
```

8. 5 GET /api/series-sugeridas/usuarios/{usuarioId}/subbloques/{subBloqueId}/ejercicios/{ejercicioId}/series

Ruta: /api/series-sugeridas/usuarios/{usuarioId}/subbloques/{subBloqueId}/ejercicios/{ejercicioId}/series

Método HTTP: GET

Descripción: Recupera las series sugeridas que un entrenador ha asignado a un usuario concreto para un ejercicio sugerido dentro de un sub-bloque. Es una consulta que filtra por usuario, sub-bloque y ejercicio sugerido.

Seguridad / Acceso:

- **Requiere token JWT en encabezado Authorization: Bearer <token>.**
- **¿El token debe corresponder al propio usuario (para ver sus series asignadas) o a un entrenador admin? Según la implementación:**
  - **En el controlador (SerieSugeridaController), se realiza:**
    1. **Validación del token (Bearer y válido).**
    2. **Usuario usuario = usuarioService.obtenerUsuarioDesdeToken(token). Sólo usuarios, no entrenadores.**
    3. **Verificar que usuario.getId().equals(usuarioId); si no, devuelve 403 Forbidden.**

4. **Llamada a serieSugeridaService.obtenerSeriesAsignadas(usuarioId, subBloqueId, ejercicioId).**

- **Por lo tanto, sólo el propio usuario puede consultar sus series asignadas; entrenadores NO usan este endpoint.**

Headers:

- **Authorization: Bearer <token>**

Path Parameters:

- **usuarioId (Long, obligatorio): ID del usuario para quien se obtienen las series asignadas.**
- **subBloqueId (Long, obligatorio): ID del sub-bloque en el que se asignaron las series.**
- **ejercicioId (Long, obligatorio): ID del ejercicio sugerido.**

Request Body: Ninguno.

Flujo interno:

1. **Validar token:**
   - **Si falta o inválido → 401 Unauthorized ("Token no proporcionado o inválido." o "Token inválido o expirado.").**
2. **Obtener usuario desde token:**
   - **Usuario usuario = usuarioService.obtenerUsuarioDesdeToken(token). Si token no corresponde a usuario (sino entrenador), obtenerUsuarioDesdeToken lanza excepción → 401 Unauthorized.**
3. **Verificar autorización:**
   - **Comparar usuario.getId() con usuarioId. Si no coinciden → 403 Forbidden ("Acceso denegado.").**
4. **Validar existencia de recurso:**
   - **El servicio serieSugeridaService.obtenerSeriesAsignadas(usuarioId, subBloqueId, ejercicioId) realizará:**
     1. **Verificar que usuarioRepository.existsById(usuarioId). Si no, lanza IllegalArgumentException("Usuario no encontrado: " + usuarioId).**

2. **Verificar que subBloqueRepository.existsById(subBloqueId). Si no, lanza IllegalArgumentException("Sub-bloque no encontrado: " + subBloqueId).**
3. **Verificar que ejercicioSugeridoRepository.existsById(ejercicioId). Si no, lanza IllegalArgumentException("Ejercicio sugerido no encontrado: " + ejercicioId).**
4. **Llamar a serieSugeridaRepository.findAssignedSeries(usuarioId, subBloqueId, ejercicioId), que implementa un JOIN para filtrar series asignadas.**
5. **Mapea cada SerieSugerida a SerieSugeridaDTO(id, numeroSerie, repeticiones, peso).**

5. **Devolver respuesta:**
   - **200 OK con cuerpo JSON: lista de series asignadas; por ejemplo:**

```
[
  { "id": 301, "numeroSerie": 1, "repeticiones": 10, "peso": 45.0 },
  { "id": 302, "numeroSerie": 2, "repeticiones": 8,  "peso": 50.0 }
]
```

   - **Si no existen series asignadas → array vacío [].**
   - **Si alguna validación falla (usuario, sub-bloque o ejercicio no existen) → 404 Not Found o 400 Bad Request con mensaje de la excepción.**
   - **Si 403 Forbidden por acceso distinto de usuario → "Acceso denegado.".**
   - **Otros errores → 500 Internal Server Error.**

Respuestas posibles:

- **200 OK: Array de series sugeridas asignadas (vacío si ninguna).**
- **401 Unauthorized: Token ausente o inválido, o token de entrenador o no corresponde a usuario.**
- **403 Forbidden: El usuario autenticado trata de acceder a datos de otro usuario.**
- **404 Not Found / 400 Bad Request:**
  - **"Usuario no encontrado: X"**
  - **"Sub-bloque no encontrado: Y"**

- “Ejercicio sugerido no encontrado: Z”

- **500 Internal Server Error: Errores imprevistos.**

Ejemplo con curl:

```
curl -X GET http://localhost:8080/api/series-sugeridas/usuarios/10/subbloques/78/ejercicios/5
  -H "Authorization: Bearer <token_usuario>"
```

9. Endpoints de Usuario (consumo de bloques/semana/sub-bloque/ejercicios)

9.1 GET /api/usuarios/{usuarioId}/bloques

Ruta: /api/usuarios/{usuarioId}/bloques

Método HTTP: GET

Descripción: Recupera la lista de bloques que están asignados al usuario con ID usuarioId.

Seguridad / Acceso:

- **Requiere token JWT en encabezado Authorization: Bearer <token>.**
- **Token debe corresponder al usuario cuyo ID es usuarioId. No está permitido que un usuario vea bloques de otro.**
- **También podría permitir que un entrenador NO admin vea los bloques de su cliente? En la implementación actual:**
    - **Se intenta usuarioService.obtenerUsuarioDesdeToken(token). Si falla (no es usuario), se intenta como entrenador; si es entrenador, se permite ver bloques de cualquier usuario: en el controlador, si tokenUsuario != null, compara IDs; si tokenEntrenador != null, permite.**

Headers:

- **Authorization: Bearer <token>**

Path Parameters:

- **usuarioId (Long, obligatorio): ID del usuario cuyas asignaciones se consultan.**

Request Body: Ninguno.

Flujo interno:

1. **Validar token:**
    - **Si falta o inválido → 401 Unauthorized.**
2. **Determinar actor:**
    - **Intentar usuarioService.obtenerUsuarioDesdeToken(token):**
        - **Si éxito, actor es usuario normal.**
        - **Si lanza excepción, intentar entrenadorService.obtenerEntrenadorDesdeToken(token):**
            - **Si éxito, actor es entrenador.**
            - **Si lanza excepción, devolver 401 Unauthorized.**
3. **Verificar autorización:**
    - **Si actor es usuario normal: tokenUsuario.getId().equals(usuarioId). Si no → 403 Forbidden.**
    - **Si actor es entrenador: se permite acceder a bloques de cualquier usuario (según implementación: "(el entrenador puede ver cualquier usuarioId)").**
4. **Llamar al servicio:**
    - **Set<BloqueUsuarioDTO> bloques = bloqueServiceUsuario.obtenerBloquesAsignados(usuarioId).**
    - **En BloqueServiceUsuarioImpl.obtenerBloquesAsignados:**

1. Busca Usuario usuario = usuarioRepository.findById(usuarioId). Si no existe → IllegalArgumentException("Usuario no encontrado").

2. Recupera bloqueRepository.findByUsuarioId(usuarioId) → lista de bloques asignados.

3. Mapea a BloqueUsuarioDTO(id, nombre).

5. **Devolver respuesta:**
    - **200 OK con array JSON:**

```
[
  { "id": 11, "nombre": "Bloque A" },
  { "id": 12, "nombre": "Bloque B" }
]
```

- **Si usuario no encontrado → 404 Not Found con mensaje "Usuario no encontrado".**
- **Si autorización falla → 403 Forbidden.**
- **Otros errores → 500 Internal Server Error.**

Respuestas posibles:

- **200 OK: Lista de bloques (vacía si no hay asignados).**
- **401 Unauthorized: Token ausente o inválido.**
- **403 Forbidden: Usuario intenta ver bloques de otro, y actor es usuario; en caso de entrenador, se permite.**
- **404 Not Found: Usuario no existe.**
- **500 Internal Server Error: Errores inesperados.**

Ejemplo con curl (usuario):

```
curl -X GET http://localhost:8080/api/usuarios/10/bloques \
  -H "Authorization: Bearer <token_usuario_10>"
```

Ejemplo con curl (entrenador):

```
curl -X GET http://localhost:8080/api/usuarios/10/bloques \
  -H "Authorization: Bearer <token_entrenador>"
```

9.2. GET /api/usuarios/{usuarioId}/bloques/{bloqueId}/semanas

Ruta: /api/usuarios/{usuarioId}/bloques/{bloqueId}/semanas

Método HTTP: GET

Descripción: Recupera la lista de semanas de un bloque asignado al usuario. Devuelve SemanaUsuarioDTO(id, numeroSemana).

Seguridad / Acceso:

- **Token JWT en Authorization: Bearer <token>.**
- **Actor puede ser:**
  - **Usuario: solo si coincide usuarioId.**
  - **Entrenador: se permite para cualquier usuarioId.**

Headers:

- **Authorization: Bearer <token>**

Path Parameters:

- **usuarioId (Long, obligatorio): ID del usuario objetivo.**
- **bloqueId (Long, obligatorio): ID del bloque cuya semanas se consultan.**

Request Body: Ninguno.

Flujo interno:

1. **Validar token: Si falta o inválido → 401 Unauthorized.**
2. **Determinar actor: Igual que en 9.1.**
3. **Verificar autorización:**
   - **Si actor es usuario normal: tokenUsuario.getId().equals(usuarioId). Si no → 403 Forbidden.**
   - **Si actor es entrenador: permite.**
4. **Validar pertenencia del bloque:**
   - **En SemanaServiceUsuarioImpl.obtenerSemanasPorBloque(bloqueId, usuarioId):**
     1. **Busca Usuario usuario = usuarioRepository.findById(usuarioId). Si no existe → IllegalArgumentException("Usuario no encontrado").**
     2. **Busca Bloque bloque = bloqueRepository.findById(bloqueId). Si no existe → IllegalArgumentException("Bloque no encontrado").**
     3. **Verificar bloque.getUsuario().getId().equals(usuario.getId()). Si no → IllegalArgumentException("Este bloque no pertenece al usuario.").**

4. **Recupera semanaRepository.findByBloque(bloque) y mapea a SemanaUsuarioDTO(id, numeroSemana).**

5. **Devolver respuesta:**

   - **200 OK con array JSON:**

```
[
  { "id": 21, "numeroSemana": 1 },
  { "id": 22, "numeroSemana": 2 }
]
```

   - **Si bloque no existe o no pertenece al usuario → 400 Bad Request o 404 Not Found con mensaje.**
   - **Si autorización falla → 403 Forbidden.**
   - **Otros errores → 500 Internal Server Error.**

Respuestas posibles:

- **200 OK: Lista de semanas (vacía si bloque sin semanas).**
- **401 Unauthorized, 403 Forbidden, 404 Not Found o 400 Bad Request ("Usuario no encontrado", "Bloque no encontrado", "Este bloque no pertenece al usuario.").**

Ejemplo con curl:

```
curl -X GET http://localhost:8080/api/usuarios/10/bloques/11/semanas \
  -H "Authorization: Bearer <token_usuario_10>"
```

9.3. GET /api/usuarios/{usuarioId}/semanas/{semanaId}/subbloques

Ruta: /api/usuarios/{usuarioId}/semanas/{semanaId}/subbloques

Método HTTP: GET

Descripción: Recupera la lista de sub-bloques de una semana asignada al usuario. Devuelve SubBloqueUsuarioDTO(id, nombre).

Seguridad / Acceso:

- **Token JWT en Authorization: Bearer <token>.**
- **Actor puede ser:**

- **Usuario: solo si coincide usuarioId.**
- **Entrenador: permite cualquier usuarioId.**

Headers:

- **Authorization: Bearer <token>**

Path Parameters:

- **usuarioId (Long, obligatorio): ID del usuario.**
- **semanaId (Long, obligatorio): ID de la semana cuyas sub-bloques se consultan.**

Request Body: Ninguno.

Flujo interno:

1. **Validar token: Si falta o inválido → 401 Unauthorized.**
2. **Determinar actor: Igual que en 9.1.**
3. **Verificar autorización:**
   - **Si usuario normal: tokenUsuario.getId().equals(usuarioId). Si no → 403 Forbidden.**
   - **Si entrenador: permite.**
4. **Validar pertenencia de la semana al usuario:**
   - **Llamada a subBloqueServiceUsuario.obtenerSubBloquesPorSemana(semanaId, usuarioId):**
     1. **Busca Usuario usuario = usuarioRepository.findById(usuarioId). Si no existe → IllegalArgumentException("Usuario no encontrado").**
     2. **Busca Semana semana = semanaRepository.findById(semanaId). Si no existe → IllegalArgumentException("Semana no encontrada").**
     3. **Bloque bloque = semana.getBloque(). Verificar bloque.getUsuario().getId().equals(usuario.getId()). Si no → IllegalArgumentException("Esta semana no pertenece al usuario.").**
     4. **Recupera subBloqueRepository.findBySemana(semana), mapea a SubBloqueUsuarioDTO(id, nombre).**

5. **Devolver respuesta:**
    - **200 OK con array JSON:**

```
[
  { "id": 31, "nombre": "Sub-bloque A" },
  { "id": 32, "nombre": "Sub-bloque B" }
]
```

    - **Si no hay sub-bloques → array vacío.**
    - **Si validación falla → 400 Bad Request o 404 Not Found con mensaje "Usuario no encontrado", "Semana no encontrada" o "Esta semana no pertenece al usuario.".**
    - **Si autorización falla → 403 Forbidden.**

Respuestas posibles:

- **200 OK: Lista de sub-bloques.**
- **401 Unauthorized, 403 Forbidden, 404 Not Found o 400 Bad Request.**

Ejemplo con curl:

```
curl -X GET http://localhost:8080/api/usuarios/10/semanas/21/subbloques \
  -H "Authorization: Bearer <token_usuario_10>"
```

9.4. GET /api/usuarios/{usuarioId}/subbloques/{subBloqueId}/ejercicios-sugeridos

Ruta: /api/usuarios/{usuarioId}/subbloques/{subBloqueId}/ejercicios-sugeridos

Método HTTP: GET

Descripción: Recupera los ejercicios sugeridos dentro de un sub-bloque asignado al usuario. Devuelve lista de EjercicioSugeridoUsuarioDTO(id, nombre, grupoMuscular).

Seguridad / Acceso:

- **Token JWT en Authorization: Bearer <token>.**
- **Actor puede ser:**
    - **Usuario: solo si coincide usuarioId.**

- **Entrenador: permite (ver ejercicios sugeridos asignados al usuario).**

Headers:

- **Authorization: Bearer <token>**

Path Parameters:

- **usuarioId (Long, obligatorio): ID del usuario.**
- **subBloqueId (Long, obligatorio): ID del sub-bloque cujos ejercicios sugeridos se consultan.**

Request Body: Ninguno.

Flujo interno:

1. **Validar token: Si falta o inválido → 401 Unauthorized.**
2. **Determinar actor: Igual que en 9.1.**
3. **Verificar autorización:**
   - **Si usuario normal: tokenUsuario.getId().equals(usuarioId). Si no → 403 Forbidden.**
   - **Si entrenador: permite.**
4. **Validar pertenencia del sub-bloque al usuario:**
   - **En ejercicioSugeridoService.obtenerEjerciciosPorSubBloque(subBloqueId) se lista todos los ejercicios de ese sub-bloque, pero hay que asegurar que el sub-bloque pertenece a la semana/bloque asignados al usuario.**
   - **En la implementación actual del controlador:**
     - **Primero valida token y usuario.**
     - **Luego directamente retorna ejercicioSugeridoService.obtenerEjerciciosPorSubBloque(subBloqueId), sin verificar que el sub-bloque pertenece al usuario.**
   - **Recomendación: antes de listar, verificar:**

1. SubBloque sb = subBloqueRepository.findById(subBloqueId). Si no existe → 404 Not Found("Subbloque no encontrado").

2. Bloque bloque = sb.getSemana().getBloque(). Verificar bloque.getUsuario().getId().equals(usuarioId). Si no, devolver 403 Forbidden("Acceso denegado: sub-bloque no pertenece al usuario.").

5. **Llamar al servicio:**
    - **Set<EjercicioSugeridoDTO> ejercicios = ejercicioSugeridoService.obtenerEjerciciosPorSubBloque(subBloqueId) .**
    - **Luego mapear a DTO de usuario si se prefiere: EjercicioSugeridoUsuarioDTO con mismo campo id, nombre, grupoMuscular.**
6. **Devolver respuesta:**
    - **200 OK con array JSON:**

```
[
  { "id": 55, "nombre": "Press de banca", "grupoMuscular": "Pecho" },
  { "id": 56, "nombre": "Fondos",        "grupoMuscular": "Tríceps" }
]
```

    - **Si no hay ejercicios sugeridos en ese sub-bloque → array vacío.**
    - **Si validaciones fallan → 404 Not Found o 403 Forbidden con mensaje.**

Respuestas posibles:

- **200 OK: Lista de ejercicios sugeridos.**
- **401 Unauthorized, 403 Forbidden, 404 Not Found o 400 Bad Request.**

Ejemplo con curl:

```
curl -X GET http://localhost:8080/api/usuarios/10/subbloques/31/ejercicios-sugeridos \
  -H "Authorization: Bearer <token_usuario_10>"
```

10. Ejercicios personales de Usuario

10.1. POST /api/usuarios/{usuarioId}/subbloques/{subBloqueId}/ejercicios

- **Ruta: /api/usuarios/{usuarioId}/subbloques/{subBloqueId}/ejercicios**
- **Método HTTP: POST**

- **Descripción: Añade un nuevo ejercicio personal para el usuario identificado por usuarioId, dentro del sub-bloque subBloqueId. El ejercicio se crea en ejercicio_usuario con datos como apiEjercicioId, nombre, grupoMuscular.**
- **Seguridad / Acceso:**
    - **Requiere token JWT en encabezado Authorization: Bearer <token>.**
    - **El token debe corresponder a un usuario ("USUARIO") cuyo ID sea igual a usuarioId. No puede un usuario crear ejercicios para otro.**
- **Headers:**
    - **Authorization: Bearer <token>**
    - **Content-Type: application/json**
- **Path Parameters:**
    - **usuarioId (Long, obligatorio): ID del usuario que crea el ejercicio.**
    - **subBloqueId (Long, obligatorio): ID del sub-bloque en el que se asigna el ejercicio.**
- **Request Body (JSON):**

```
{
  "apiEjercicioId": "ext123",
  "nombre": "Press de banca inclinado",
  "grupoMuscular": "Pecho"
}
```

- **apiEjercicioId (String, obligatorio): Identificador externo o propio del ejercicio.**
- **nombre (String, obligatorio): Nombre descriptivo del ejercicio.**
- **grupoMuscular (String, opcional u obligatorio según requisitos): Grupo muscular asociado.**

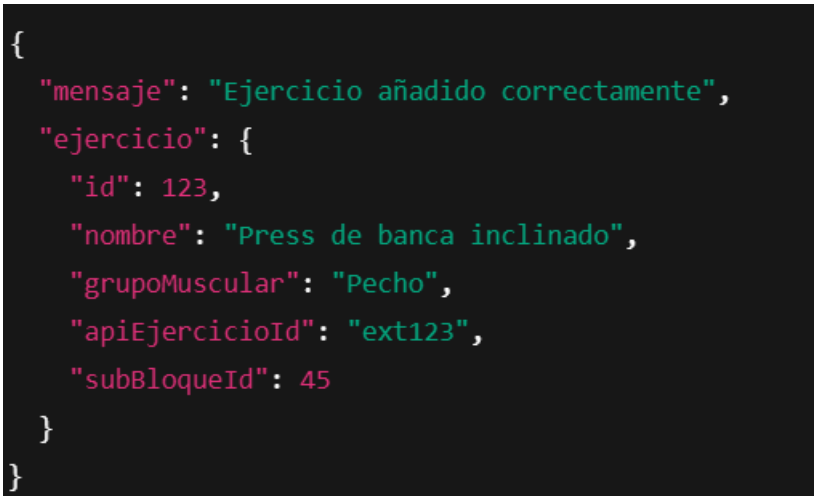
Flujo interno:

1. **Validar token:**
    - **Si falta o no empieza con Bearer → responde 401 Unauthorized con mensaje "Token no proporcionado o inválido.".**
    - **Extraer token y validar con jwtUtil.validateToken(token). Si inválido o expirado → 401 Unauthorized con "Token inválido o expirado.".**

2. **Obtener usuario del token:**
    - **Usuario usuarioToken = usuarioService.obtenerUsuarioDesdeToken(token). Si no corresponde a usuario válido → lanza excepción y se captura devolviendo 401 Unauthorized.**
3. **Verificar autorización:**
    - **Comparar usuarioToken.getId() con usuarioId de la ruta. Si no coinciden → 403 Forbidden con "No puedes modificar ejercicios de otro usuario.".**
4. **Validar existencia de sub-bloque:**
    - **SubBloque subBloque = subBloqueRepository.findById(subBloqueId). Si no existe → lanza IllegalArgumentException("Subbloque no encontrado"), devolver 400 Bad Request o 404 Not Found con este mensaje.**
    - **Además, verificar que el sub-bloque pertenece al usuario:**
        - **Opcionalmente: Bloque bloque = subBloque.getSemana().getBloque() y luego bloque.getUsuario().getId().equals(usuarioId). Si no, 403 Forbidden con "El sub-bloque no pertenece al usuario.". En el controlador actual, no se hace explícitamente, pero es recomendable añadir esta validación.**
5. **Crear ejercicio:**
    - **Construir EjercicioUsuarioRequestDTO dto con datos del body.**
    - **Llamar a ejercicioUsuarioService.agregarEjercicio(subBloqueId, dto, usuarioId):**
        - **En el servicio:**
            1. **Buscar Usuario usuario por usuarioId. Si no existe → lanza IllegalArgumentException("Usuario no encontrado").**
            2. **Buscar SubBloque subBloque por subBloqueId. Si no existe → lanza IllegalArgumentException("Subbloque no encontrado").**

3. **Crear nueva entidad EjercicioUsuario ejercicio, setear nombre, grupoMuscular, apiEjercicioId, asociar usuario y subBloque.**
4. **Guardar: ejercicioUsuarioRepository.save(ejercicio).**
5. **Mapear a EjercicioUsuarioDTO(id, nombre, grupoMuscular, apiEjercicioId, subBloqueId).**

6. **Devolver respuesta:**
    - **En caso de éxito: 201 Created con cuerpo JSON:**



```
{
  "mensaje": "Ejercicio añadido correctamente",
  "ejercicio": {
    "id": 123,
    "nombre": "Press de banca inclinado",
    "grupoMuscular": "Pecho",
    "apiEjercicioId": "ext123",
    "subBloqueId": 45
  }
}
```

    - **Si IllegalArgumentException por usuario o sub-bloque no encontrado o acceso denegado → 400 Bad Request o 404 Not Found según convención, con mensaje descriptivo.**
    - **Si autorización falla → 403 Forbidden.**
    - **Otros errores → 500 Internal Server Error.**

Respuestas posibles:

- **201 Created: Ejercicio creado, devuelve DTO.**
- **400 Bad Request: Datos inválidos o sub-bloque/usuario no encontrado.**
- **401 Unauthorized: Token ausente o inválido.**
- **403 Forbidden: Intento de crear ejercicio para otro usuario o sub-bloque no perteneciente.**
- **500 Internal Server Error: Errores imprevistos.**

Ejemplo con curl:

```
curl -X POST http://localhost:8080/api/usuarios/10/subbloques/45/ejercicios \
  -H "Authorization: Bearer <token_usuario_10>" \
  -H "Content-Type: application/json" \
  -d '{"apiEjercicioId":"ext123","nombre":"Press de banca inclinado","grupoMuscular":"Pecho"}
```

10.2. GET /api/usuarios/{usuarioId}/subbloques/{subBloqueId}/ejercicios

- **Ruta: /api/usuarios/{usuarioId}/subbloques/{subBloqueId}/ejercicios**
- **Método HTTP: GET**
- **Descripción: Lista los ejercicios personales que el usuario identificador usuarioId ha añadido dentro del sub-bloque subBloqueId.**
- **Seguridad / Acceso:**
    - **Token JWT en Authorization: Bearer <token>.**
    - **El token debe corresponder al usuario cuyo ID es usuarioId. No se permite listar ejercicios de otro usuario.**
    - **Un entrenador (rol "ADMIN") no utiliza este endpoint; si llama, usuarioService.obtenerUsuarioDesdeToken(token) lanzará excepción, a menos que se amplíe la lógica para permitir entrenadores ver ejercicios de sus clientes. Actualmente solo usuario.**
- **Headers:**
    - **Authorization: Bearer <token>**
- **Path Parameters:**
    - **usuarioId (Long, obligatorio): ID del usuario.**
    - **subBloqueId (Long, obligatorio): ID del sub-bloque.**
- **Request Body: Ninguno.**
- **Flujo interno:**

1. Validar token: Si falta o inválido → 401 Unauthorized.

2. Obtener usuario del token: Usuario usuarioToken = usuarioService.obtenerUsuarioDesdeToken(token). Si no es usuario válido → 401 Unauthorized.

3. **Verificar autorización: usuarioToken.getId().equals(usuarioId). Si no → 403 Forbidden.**

4. **Validar pertenencia sub-bloque:**

    - **SubBloque subBloque = subBloqueRepository.findById(subBloqueId). Si no existe → 404 Not Found("Subbloque no encontrado").**
    - **Verificar que subBloque.getSemana().getBloque().getUsuario().getId().equals(usuarioId). Si no → 403 Forbidden("Acceso denegado: sub-bloque no pertenece al usuario.").**

5. **Llamar al servicio:**

    - **Set<EjercicioUsuarioDTO> ejercicios = ejercicioUsuarioService.obtenerEjerciciosPorSubBloque(subBloqueId, usuarioId).**
    - **En el servicio: findByUsuarioIdAndSubBloqueId(usuarioId, subBloqueId), mapear a DTO.**

6. **Devolver respuesta:**

    - **200 OK con array JSON:**


```
[
  {
    "id": 123,
    "nombre": "Press de banca inclinado",
    "grupoMuscular": "Pecho",
    "apiEjercicioId": "ext123",
    "subBloqueId": 45
  },
  {
    "id": 124,
    "nombre": "Curl de bíceps",
    "grupoMuscular": "Bíceps",
    "apiEjercicioId": "ext456",
    "subBloqueId": 45
  }
]
```


    - **Si no hay ejercicios, array vacío [].**

- Si validaciones fallan → 404 Not Found o 403 Forbidden con mensaje.

Respuestas posibles:

- **200 OK: Lista de ejercicios.**
- **401 Unauthorized: Token ausente o inválido.**
- **403 Forbidden: Acceso a recursos de otro usuario o sub-bloque no perteneciente.**
- **404 Not Found: Sub-bloque no existe o usuario no existe (verificado en servicio).**
- **500 Internal Server Error.**

Ejemplo con curl:

```
curl -X GET http://localhost:8080/api/usuarios/10/subbloques/45/ejercicios \
  -H "Authorization: Bearer <token_usuario_10>"
```

10.3. DELETE /api/usuarios/{usuarioId}/subbloques/{subBloqueId}/ejercicios/{ejercicioId}

- **Ruta: /api/usuarios/{usuarioId}/subbloques/{subBloqueId}/ejercicios/{ejercicioId}**
- **Método HTTP: DELETE**
- **Descripción: Elimina un ejercicio personal identificado por ejercicioId, siempre que pertenezca al usuario usuarioId y al sub-bloque subBloqueId.**
- **Seguridad / Acceso:**
  - **Token JWT en Authorization: Bearer <token>.**
  - **El token debe corresponder al usuario cuyo ID es usuarioId.**
- **Headers:**
  - **Authorization: Bearer <token>**
- **Path Parameters:**
  - **usuarioId (Long, obligatorio): ID del usuario.**
  - **subBloqueId (Long, obligatorio): ID del sub-bloque.**
  - **ejercicioId (Long, obligatorio): ID del ejercicio a eliminar.**

- **Request Body: Ninguno.**
- **Flujo interno:**

1. **Validar token: Si falta o inválido → 401 Unauthorized.**
2. **Obtener usuario del token: Usuario usuarioToken = usuarioService.obtenerUsuarioDesdeToken(token). Si no es usuario válido → 401 Unauthorized.**
3. **Verificar autorización: usuarioToken.getId().equals(usuarioId). Si no → 403 Forbidden.**
4. **Validar existencia y pertenencia del ejercicio:**
    - **EjercicioUsuarioDTO dto = ejercicioUsuarioService.obtenerEjercicio(ejercicioId, usuarioId). En el servicio:**
        1. **Busca EjercicioUsuario ejercicio = ejercicioUsuarioRepository.findById(ejercicioUsuarioId). Si no existe → lanza IllegalArgumentException("Ejercicio no encontrado").**
        2. **Verifica ejercicio.getUsuario().getId().equals(usuarioId). Si no → lanza IllegalArgumentException("Acceso denegado.").**
        3. **Mapea a DTO con subBloqueId asociado.**
    - **En el controlador, además comparar dto.getSubBloqueId().equals(subBloqueId). Si no coincide → 400 Bad Request con "El ejercicio no pertenece al sub-bloque indicado.".**
5. **Eliminar ejercicio:**
    - **ejercicioUsuarioService.eliminarEjercicio(ejercicioId, usuarioId):**
        1. **Busca EjercicioUsuario ejercicio y verifica usuario como antes.**
        2. **ejercicioUsuarioRepository.delete(ejercicio).**
6. **Devolver respuesta:**
    - **200 OK con cuerpo:**

```
"Ejercicio eliminado correctamente."
```

Si ejercicio no encontrado o no pertenece → 400 Bad Request o 404 Not Found con mensaje.

Si autorización falla → 403 Forbidden.

Otros errores → 500 Internal Server Error.

Respuestas posibles:

- **200 OK: Confirmación de eliminación.**
- **400 Bad Request: Ejercicio no pertenece al sub-bloque o mensaje de servicio.**
- **401 Unauthorized: Token ausente o inválido.**
- **403 Forbidden: Intento de borrar ejercicio de otro usuario.**
- **404 Not Found: Ejercicio o usuario no encontrado.**
- **500 Internal Server Error.**

Ejemplo con curl:

```
curl -X DELETE http://localhost:8080/api/usuarios/10/subbloques/45/ejercicios/123 \
  -H "Authorization: Bearer <token_usuario_10>"
```

11. Series de entrenamiento de Usuario

11.1. GET /api/usuarios/{usuarioId}/ejercicios/{ejercicioId}/series

- **Ruta: /api/usuarios/{usuarioId}/ejercicios/{ejercicioId}/series**
- **Método HTTP: GET**
- **Descripción: Recupera todas las series de entrenamiento que el usuario usuarioId ha registrado para el ejercicio personal ejercicioId.**
- **Seguridad / Acceso:**
  - **Token JWT en Authorization: Bearer <token>.**
  - **El token debe corresponder al usuario con ID igual a usuarioId.**

- **Headers:**
  - **Authorization: Bearer <token>**
- **Path Parameters:**
  - **usuarioId (Long, obligatorio): ID del usuario.**
  - **ejercicioId (Long, obligatorio): ID del ejercicio personal (ejercicio_usuario) cuyas series se listan.**
- **Request Body: Ninguno.**
- **Flujo interno:**

1. Validar token: Si falta o inválido → 401 Unauthorized.

2. Obtener usuario del token: Usuario usuario = usuarioService.obtenerUsuarioDesdeToken(token). Si no es usuario válido → 401 Unauthorized.

3. Verificar autorización: usuario.getId().equals(usuarioId). Si no → 403 Forbidden.

4. Validar existencia y pertenencia del ejercicio:

- **Llamar a serieEntrenamientoService.obtenerSeries(ejercicioId, usuarioId):**
  1. **Busca EjercicioUsuario ejercicio = ejercicioRepository.findById(ejercicioUsuarioId). Si no existe → lanza IllegalArgumentException("Ejercicio no encontrado").**
  2. **Verifica ejercicio.getUsuario().getId().equals(usuarioId). Si no → lanza IllegalArgumentException("Acceso denegado").**

3. **Recupera serieRepository.findByEjercicioUsuario(ejercicio) → Set<SerieEntrenamiento>.**
4. **Mapear cada SerieEntrenamiento a SerieEntrenamientoDTO(id, numeroSerie, repeticiones, peso, esfuerzoPercibido, fecha).**

5. **Devolver respuesta:**

- **200 OK con array JSON:**


```
[
  {
    "id": 501,
    "numeroSerie": 1,
    "repeticiones": 10,
    "peso": 50.0,
    "esfuerzoPercibido": 7,
    "fecha": "2025-06-01"
  },
  {
    "id": 502,
    "numeroSerie": 2,
    "repeticiones": 8,
    "peso": 55.0,
    "esfuerzoPercibido": 8,
    "fecha": "2025-06-01"
  }
]
```


- **Si no hay series → array vacío [].**
- **Si ejercicio no existe o no pertenece al usuario → 400 Bad Request o 404 Not Found con mensaje "Ejercicio no encontrado" o "Acceso denegado".**

Respuestas posibles:

- **200 OK: Lista de SerieEntrenamientoDTO.**

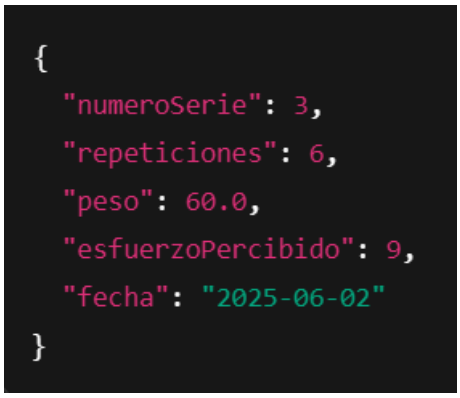
- **401 Unauthorized: Token ausente o inválido.**
- **403 Forbidden: Intento de listar series de otro usuario.**
- **404 Not Found o 400 Bad Request: Ejercicio no encontrado o acceso denegado.**
- **500 Internal Server Error.**

Ejemplo con curl:

```
curl -X GET http://localhost:8080/api/usuarios/10/ejercicios/123/series \
  -H "Authorization: Bearer <token_usuario_10>"
```

11.2. POST /api/usuarios/{usuarioId}/ejercicios/{ejercicioId}/series

- **Ruta: /api/usuarios/{usuarioId}/ejercicios/{ejercicioId}/series**
- **Método HTTP: POST**
- **Descripción: Crea una nueva serie de entrenamiento para el ejercicio personal ejercicioId del usuario usuarioId. Se incluyen campos como numeroSerie, repeticiones, peso, esfuerzoPercibido y fecha.**
- **Seguridad / Acceso:**
  - **Token JWT en Authorization: Bearer <token>.**
  - **El token debe corresponder al usuario cuyo ID es usuarioId.**
- **Headers:**
  - **Authorization: Bearer <token>**
  - **Content-Type: application/json**
- **Path Parameters:**
  - **usuarioId (Long, obligatorio): ID del usuario.**
  - **ejercicioId (Long, obligatorio): ID del ejercicio personal al que se añade la serie.**
- **Request Body (JSON):**



- **numeroSerie (int, obligatorio): Número secuencial de la serie.**
- **repeticiones (Integer, opcional u obligatorio): Repeticiones recomendadas.**
- **peso (Double, opcional u obligatorio): Peso levantado o sugerido.**
- **esfuerzoPercibido (Integer, opcional u obligatorio): Escala de esfuerzo.**
- **fecha (LocalDate, obligatorio): Fecha de la serie.**

Flujo interno:

1. **Validar token: Si falta o inválido → 401 Unauthorized.**
2. **Obtener usuario del token: Usuario usuario = usuarioService.obtenerUsuarioDesdeToken(token). Si no es usuario válido → 401 Unauthorized.**
3. **Verificar autorización: usuario.getId().equals(usuarioId). Si no → 403 Forbidden.**
4. **Validar existencia y pertenencia del ejercicio:**
    - **En serieEntrenamientoService.agregarSerie(ejercicioId, dto, usuarioId):**
        1. **Busca EjercicioUsuario ejercicio = ejercicioRepository.findById(ejercicioUsuarioId). Si no existe → lanza IllegalArgumentException("Ejercicio no encontrado").**

2. **Verifica ejercicio.getUsuario().getId().equals(usuarioId). Si no → lanza IllegalArgumentException("Acceso denegado").**

5. **Determinar sub-bloque asociado:**
    - **Según la implementación, se obtiene subBloqueId vía: serieRepository.findFirstSubBloqueIdByUsuarioId(usuarioId). Esto busca el primer sub-bloque asociado al usuario; es una lógica implícita.**
    - **Verificar que tal sub-bloque existe: SubBloque subBloque = subBloqueRepository.findById(subBloqueId).**
    - **Consideración: Podría quererse que el cliente especifique explícitamente sub-bloque en futuro.**
6. **Crear entidad SerieEntrenamiento:**
    - **serie.setEjercicioUsuario(ejercicio).**
    - **serie.setSubBloque(subBloque).**
    - **Asignar campos numeroSerie, repeticiones, peso, esfuerzoPercibido, fecha.**
    - **Guardar: serieRepository.save(serie).**
7. **Mapear a DTO:**
    - **SerieEntrenamientoDTO nuevaDTO = toDTO(guardada).**
8. **Devolver respuesta:**
    - **201 Created con cuerpo JSON de la serie creada:**

```
{
  "id": 502,
  "numeroSerie": 3,
  "repeticiones": 6,
  "peso": 60.0,
  "esfuerzoPercibido": 9,
  "fecha": "2025-06-02"
}
```


- **Si ejercicio no existe o no pertenece → 400 Bad Request o 404 Not Found con mensaje.**
- **Si token o autorización falla → 401 Unauthorized / 403 Forbidden.**
- **Otros errores → 500 Internal Server Error.**

Respuestas posibles:

- **201 Created: DTO de la serie creada.**
- **400 Bad Request: Ejercicio no encontrado o acceso denegado.**
- **401 Unauthorized, 403 Forbidden, 500 Internal Server Error.**

Ejemplo con curl:

```
curl -X POST http://localhost:8080/api/usuarios/10/ejercicios/123/series \
  -H "Authorization: Bearer <token_usuario_10>" \
  -H "Content-Type: application/json" \
  -d '{"numeroSerie":3,"repeticiones":6,"peso":60.0,"esfuerzoPercibido":9,"fecha":"2025-06-02"
```

11.3. PUT /api/usuarios/{usuarioId}/series/{serieId}

- **Ruta: /api/usuarios/{usuarioId}/series/{serieId}**
- **Método HTTP: PUT**
- **Descripción: Actualiza una serie de entrenamiento existente con ID serieId, modificando campos como numeroSerie, repeticiones, peso, esfuerzoPercibido, fecha.**
- **Seguridad / Acceso:**
  - **Token JWT en Authorization: Bearer <token>.**

- El token debe corresponder al usuario cuyo ID es usuarioId.

- **Headers:**
  - **Authorization: Bearer <token>**
  - **Content-Type: application/json**
- **Path Parameters:**
  - **usuarioId (Long, obligatorio): ID del usuario.**
  - **serieId (Long, obligatorio): ID de la serie que se actualiza.**
- **Request Body (JSON):**

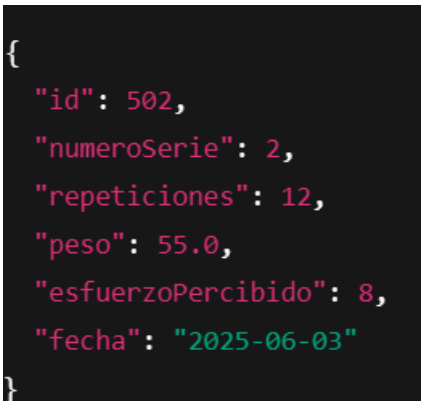
```
{
  "numeroSerie": 2,
  "repeticiones": 12,
  "peso": 55.0,
  "esfuerzoPercibido": 8,
  "fecha": "2025-06-03"
}
```

Flujo interno:

1. **Validar token: Si falta o inválido → 401 Unauthorized.**
2. **Obtener usuario del token: Usuario usuario = usuarioService.obtenerUsuarioDesdeToken(token). Si no → 401 Unauthorized.**
3. **Verificar autorización: usuario.getId().equals(usuarioId). Si no → 403 Forbidden.**
4. **Validar existencia y pertenencia de la serie:**
   - **Llamar a serieEntrenamientoService.actualizarSerie(serieId, dto, usuarioId):**
     1. **Busca SerieEntrenamiento serie = serieRepository.findById(serieId). Si no existe → lanza IllegalArgumentException("Serie no encontrada").**

2. **Verifica serie.getEjercicioUsuario().getUsuario().getId().equals(usuarioId). Si no → lanza IllegalArgumentException("Acceso denegado").**

5. **Actualizar campos:**
   - **En el servicio: serie.setNumeroSerie(dto.getNumeroSerie()), etc., incluyendo fecha.**
   - **Guardar: serieRepository.save(serie).**
   - **Mapear a DTO y devolver.**
6. **Devolver respuesta:**
   - **200 OK con cuerpo JSON de la serie actualizada:**



```
{
  "id": 502,
  "numeroSerie": 2,
  "repeticiones": 12,
  "peso": 55.0,
  "esfuerzoPercibido": 8,
  "fecha": "2025-06-03"
}
```

   - **Si serie no existe o no pertenece → 404 Not Found o 400 Bad Request con mensaje.**
   - **Si token o autorización falla → 401 Unauthorized / 403 Forbidden.**
   - **Otros errores → 500 Internal Server Error.**

Respuestas posibles:

- **200 OK: DTO actualizado.**
- **400 Bad Request o 404 Not Found: Serie no encontrada o acceso denegado.**
- **401 Unauthorized, 403 Forbidden, 500 Internal Server Error.**

Ejemplo con curl:

```
curl -X PUT http://localhost:8080/api/usuarios/10/series/502 \
  -H "Authorization: Bearer <token_usuario_10>" \
  -H "Content-Type: application/json" \
  -d '{"numeroSerie":2,"repeticiones":12,"peso":55.0,"esfuerzoPercibido":8,"fecha":"2025-06-0
```

11.4. DELETE /api/usuarios/{usuarioId}/series/{serieId}

- **Ruta: /api/usuarios/{usuarioId}/series/{serieId}**
- **Método HTTP: DELETE**
- **Descripción: Elimina la serie de entrenamiento con ID serieId, siempre que pertenezca al usuario usuarioId.**
- **Seguridad / Acceso:**
  - **Token JWT en Authorization: Bearer <token>.**
  - **El token debe corresponder al usuario cuyo ID coincide con usuarioId.**
- **Headers:**
  - **Authorization: Bearer <token>**
- **Path Parameters:**
  - **usuarioId (Long, obligatorio): ID del usuario.**
  - **serieId (Long, obligatorio): ID de la serie a eliminar.**
- **Request Body: Ninguno.**
- **Flujo interno:**

1. **Validar token: Si falta o inválido → 401 Unauthorized.**
2. **Obtener usuario del token: Usuario usuario = usuarioService.obtenerUsuarioDesdeToken(token). Si no → 401 Unauthorized.**
3. **Verificar autorización: usuario.getId().equals(usuarioId). Si no → 403 Forbidden.**

4. **Validar existencia y pertenencia de la serie:**
    - **Llamar a serieEntrenamientoService.eliminarSerie(serieId, usuarioId):**
        1. **Busca SerieEntrenamiento serie = serieRepository.findById(serieId). Si no existe → lanza IllegalArgumentException("Serie no encontrada").**
        2. **Verifica serie.getEjercicioUsuario().getUsuario().getId().equals(usuarioId). Si no → lanza IllegalArgumentException("Acceso denegado").**
        3. **serieRepository.delete(serie).**
5. **Devolver respuesta:**
    - **200 OK con cuerpo:**

```
"Serie eliminada correctamente."
```

    - **Si serie no encontrada o acceso denegado → 400 Bad Request o 404 Not Found con mensaje.**
    - **Si token o autorización falla → 401 Unauthorized / 403 Forbidden.**
    - **Otros errores → 500 Internal Server Error.**

Respuestas posibles:

- **200 OK: Confirmación de eliminación.**
- **400 Bad Request o 404 Not Found: Serie no encontrada o acceso denegado.**
- **401 Unauthorized, 403 Forbidden, 500 Internal Server Error.**

Ejemplo con curl:

```
curl -X DELETE http://localhost:8080/api/usuarios/10/series/502 \
  -H "Authorization: Bearer <token_usuario_10>"
```

12. Gráficas / Progreso

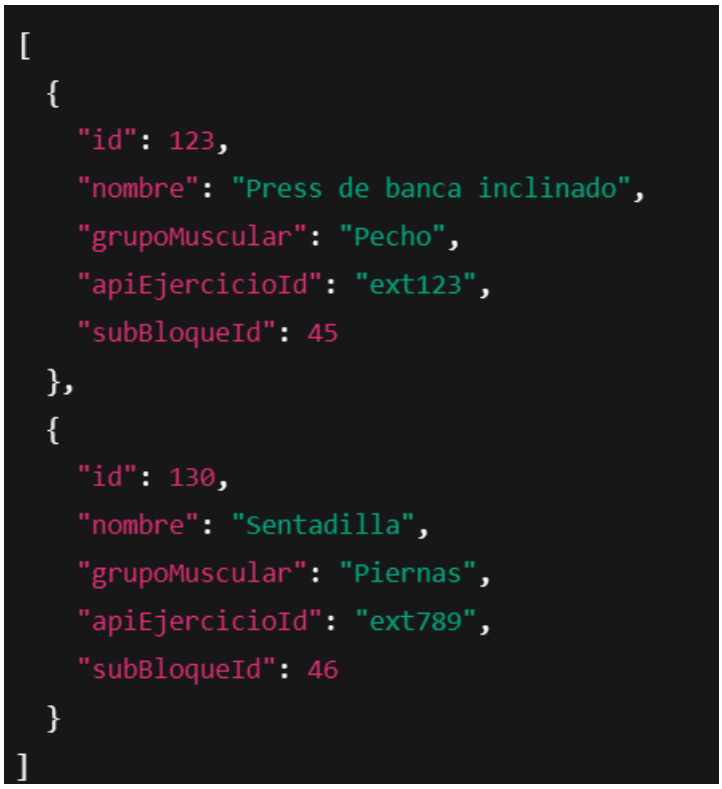
12.1. GET /api/grafica/{usuarioId}/ejercicios

- **Ruta: /api/grafica/{usuarioId}/ejercicios**
- **Método HTTP: GET**
- **Descripción: Recupera todos los ejercicios personales del usuario (ejercicio_usuario), sin duplicados por apiEjercicioId. Útil para poblar un selector en la UI de gráficas.**
- **Seguridad / Acceso:**
  - **Token JWT en Authorization: Bearer <token>.**
  - **Si el token es de un usuario: solo puede consultar su propio usuarioId.**
  - **Si el token es de un entrenador admin: se permite consultar los ejercicios de cualquier usuario. Recomendación: restringir para que el entrenador solo acceda a sus clientes; implementar verificación adicional usuario.getEntrenador().getId().equals(entrenador.getId()).**
- **Headers:**
  - **Authorization: Bearer <token>**
- **Path Parameters:**
  - **usuarioId (Long, obligatorio): ID del usuario cuyo listado de ejercicios se solicita.**
- **Request Body: Ninguno.**
- **Flujo interno:**

1. **Validar token: Si falta o inválido → 401 Unauthorized.**

2. **Determinar actor:**
    - **Intentar usuarioService.obtenerUsuarioDesdeToken(token): si éxito, actor es usuario.**
    - **Si lanza excepción, intentar entrenadorService.obtenerEntrenadorDesdeToken(token): si éxito, actor es entrenador.**
    - **Si ambos fallan → 401 Unauthorized con "Token inválido o usuario no encontrado".**
3. **Verificar autorización:**
    - **Si actor es usuario: usuario.getId().equals(usuarioId). Si no → 403 Forbidden("Acceso denegado").**
    - **Si actor es entrenador: permite, idealmente verificar que el usuario solicitado es cliente del entrenador.**
4. **Llamar al servicio:**
    - **Set<EjercicioUsuarioDTO> ejercicios = ejercicioUsuarioService.obtenerEjerciciosDeUsuario(usuarioId).**
    - **En el servicio:**
        1. **Set<EjercicioUsuario> todos = ejercicioUsuarioRepository.findByUsuarioId(usuarioId).**
        2. **Construir un Map<String, EjercicioUsuario> para eliminar duplicados según apiEjercicioId, quedando una instancia por cada identificador externo.**
        3. **Mapear a EjercicioUsuarioDTO(id, nombre, grupoMuscular, apiEjercicioId, subBloqueId).**
5. **Devolver respuesta:**

- **200 OK con array JSON:**



```
[
  {
    "id": 123,
    "nombre": "Press de banca inclinado",
    "grupoMuscular": "Pecho",
    "apiEjercicioId": "ext123",
    "subBloqueId": 45
  },
  {
    "id": 130,
    "nombre": "Sentadilla",
    "grupoMuscular": "Piernas",
    "apiEjercicioId": "ext789",
    "subBloqueId": 46
  }
]
```

  - **Si no hay ejercicios → array vacío [].**
  - **Si usuario no existe → 404 Not Found("Usuario no encontrado").**
  - **Si autorización falla → 403 Forbidden.**

Respuestas posibles:

- **200 OK: Lista de EjercicioUsuarioDTO.**
- **401 Unauthorized: Token ausente o inválido.**
- **403 Forbidden: Usuario trata de acceder a otro, o entrenador no autorizado para ese usuario.**
- **404 Not Found: Usuario no existe.**
- **500 Internal Server Error.**

Ejemplo con curl:

```
curl -X GET http://localhost:8080/api/grafica/10/ejercicios \
  -H "Authorization: Bearer <token_usuario_10>"
```

O, como entrenador:

```
curl -X GET http://localhost:8080/api/grafica/10/ejercicios \
  -H "Authorization: Bearer <token_entrenador>"
```

12.2. GET /api/grafica/{usuarioId}/ejercicios/{ejercicioUsuarioId}/max-pesos

- **Ruta: /api/grafica/{usuarioId}/ejercicios/{ejercicioUsuarioId}/max-pesos**
- **Método HTTP: GET**
- **Descripción: Para el ejercicio personal identificado por ejercicioUsuarioId, obtiene el peso máximo levantado por fecha de entrenamiento, devolviendo una serie de pares (fecha, pesoMaximo). Útil para graficar la evolución del peso en el tiempo.**
- **Seguridad / Acceso:**
  - **Token JWT en Authorization: Bearer <token>.**
  - **Si token es de usuario: solo puede consultar su propio usuarioId.**
  - **Si token es de entrenador admin: se permite, idealmente restringir a clientes.**
- **Headers:**
  - **Authorization: Bearer <token>**
- **Path Parameters:**
  - **usuarioId (Long, obligatorio): ID del usuario.**
  - **ejercicioUsuarioId (Long, obligatorio): ID del ejercicio personal.**

- **Request Body: Ninguno.**
- **Flujo interno:**

1. Validar token: Si falta o inválido → 401 Unauthorized con “Token no proporcionado.” o “Token inválido o expirado.”.

2. Determinar actor:

- **Intentar usuarioService.obtenerUsuarioDesdeToken(token): éxito → actor usuario.**
- **Si lanza excepción, intentar entrenadorService.obtenerEntrenadorDesdeToken(token): éxito → actor entrenador.**
- **Si ambos fallan → 401 Unauthorized("Token inválido o usuario no encontrado.").**

3. Verificar autorización:

- **Si actor es usuario: usuario.getId().equals(usuarioId). Si no → 403 Forbidden("Acceso denegado para este usuario.").**
- **Si actor es entrenador: permite, idealmente chequear que usuarioId corresponde a un cliente del entrenador.**

4. Llamar al servicio:

- **Set<MaxPesoDTO> resultado = serieEntrenamientoService.obtenerMaxPesoPorEjercicioUsuario(ejercicioUsuarioId).**
- **En el servicio:**
  1. **Verificar existencia de EjercicioUsuario con ID ejercicioUsuarioId: Optional<EjercicioUsuario> optEj = ejercicioRepository.findById(ejercicioUsuarioId).**

Si vacío → devuelve Collections.emptySet() según implementación actual.

- **Nota: En lugar de error, la implementación devuelve Set vacío; conviene documentar este comportamiento: si el ejercicio no existe, la respuesta es 200 OK con array vacío. Alternativamente, se podría preferir 404 Not Found.**

2. **Llamar a serieRepository.findMaxPesoByEjercicioUsuarioId(ejercicioUsuarioId), que ejecuta:**

```
SELECT new com.fittrack.Dto.MaxPesoDTO(s.fecha, MAX(s.peso))
FROM SerieEntrenamiento s
WHERE s.ejercicioUsuario.id = :ejercicioUsuarioId
  AND s.peso IS NOT NULL
GROUP BY s.fecha
```

1. **Devuelve Set<MaxPesoDTO> con pares (fecha, pesoMaximo).**

Devolver respuesta:

- **Caso ejercicio existente:**
  - **Si hay registros de series con peso → devuelve lista de objetos MaxPesoDTO:**

```
[
  { "fecha": "2025-06-01", "pesoMaximo": 55.0 },
  { "fecha": "2025-06-05", "pesoMaximo": 57.5 },
  ...
]
```

  -
    - **Si no hay series o todos sin peso → array vacío [].**
    - **Código: 200 OK.**

- **Caso ejercicio no existe:**
  - **Implementación actual: devuelve 200 OK con []. Documentar este comportamiento o ajustar a 404 Not Found.**
- **Si autorización falla → 403 Forbidden.**
- **Si token inválido → 401 Unauthorized.**
- **Otros errores → 500 Internal Server Error.**

Respuestas posibles:

- **200 OK: Array de MaxPesoDTO. Puede estar vacío tanto si no hay series como si el ejercicio no existe.**
- **401 Unauthorized: Token ausente o inválido.**
- **403 Forbidden: Usuario trata de consultar datos de otro usuario.**
- **500 Internal Server Error.**

Ejemplo con curl:

```
curl -X GET http://localhost:8080/api/grafica/10/ejercicios/123/max-pesos \
  -H "Authorization: Bearer <token_usuario_10>"
```

O como entrenador:

```
curl -X GET http://localhost:8080/api/grafica/10/ejercicios/123/max-pesos \
  -H "Authorization: Bearer <token_entrenador>"
```

13. Manejo de errores y respuestas comunes

13.1. Principios generales

1. **Uniformidad de formato**
   - **Todas las respuestas de error deben compartir un formato JSON consistente, de modo que el cliente pueda parsear fácilmente y mostrar mensajes.**

- Evitar devolver directamente cadenas sueltas; en su lugar, usar un objeto con clave estándar, por ejemplo:

```
{
  "error": "Descripción clara del error"
}
```

Opcionalmente se puede incluir un código interno o identificador (code) para catálogos de errores, p.ej.:

```
{
  "error": "Recurso no encontrado",
  "code": "USER_NOT_FOUND"
}
```

Códigos HTTP apropiados

- 400 Bad Request: Solicitud mal formada o datos inválidos (por ejemplo, JSON con campos faltantes, formatos erróneos, validaciones de negocio no cumplidas).
- 401 Unauthorized: Token ausente o inválido/expirado.
- 403 Forbidden: Usuario autenticado, pero sin permiso sobre el recurso solicitado.
- 404 Not Found: Recurso no existe (usuario, sub-bloque, ejercicio, etc.).
    - Nota: en ciertas implementaciones actuales, al consultar progreso de ejercicio inexistente, se devuelve 200 con array vacío. Se recomienda documentar dicho comportamiento o cambiar a 404 según criterio.
- 409 Conflict: Conflictos de estado (p. ej., intentar crear un recurso duplicado si se decide prevenir duplicados).
- 500 Internal Server Error: Errores imprevistos en el servidor.

Mensajes claros y específicos

- Incluir en el mensaje qué recurso o parámetro causó el error.

- **Evitar exponer detalles sensibles (stack traces); en entorno de producción, registrar internamente el stack trace, pero devolver al cliente solo el mensaje necesario.**

Centralizar el manejo de excepciones

- **Utilizar un @ControllerAdvice con métodos @ExceptionHandler para capturar excepciones comunes (por ejemplo, IllegalArgumentException, excepciones personalizadas, JwtException) y mapearlas a respuestas HTTP con JSON uniforme.**
- **Ejemplo de estructura de un @ControllerAdvice:**

```java
@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(IllegalArgumentException.class)
    public ResponseEntity<Map<String, String>> handleIllegalArg(IllegalArgumentExceptior
        // Podríamos diferenciar mensajes "no encontrado" vs "acceso denegado":
        String msg = ex.getMessage();
        HttpStatus status = HttpStatus.BAD_REQUEST;
        if (msg != null && msg.toLowerCase().contains("no encontrado")) {
            status = HttpStatus.NOT_FOUND;
        } else if (msg != null && msg.toLowerCase().contains("acceso denegado")) {
            status = HttpStatus.FORBIDDEN;
        }
        Map<String, String> body = Map.of("error", msg);
        return ResponseEntity.status(status).body(body);
    }

    @ExceptionHandler(AuthenticationException.class)
    public ResponseEntity<Map<String, String>> handleAuth(AuthenticationException ex) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
                .body(Map.of("error", ex.getMessage()));
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity<Map<String, String>> handleOther(Exception ex) {
        // Registrar stack trace internamente
        // logger.error("Error interno", ex);
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
                .body(Map.of("error", "Error interno del servidor"));
    }
}
```

De esta forma, en los controladores se puede lanzar IllegalArgumentException("Usuario no encontrado") o similar, y el handler se encarga de la respuesta adecuada.

Errores de validación de datos de entrada

- **Para requests con cuerpo JSON, especialmente en creación/actualización, validar que los campos obligatorios estén presentes y cumplan formato (p.ej., fechas con formato YYYY-MM-DD, valores numéricos dentro de rangos razonables).**
- **Se puede usar validaciones de Bean Validation (@Valid, anotaciones @NotNull, @Size, etc.) en los DTOs de request. Cuando se produce una violación, Spring lanza MethodArgumentNotValidException, que se puede capturar en @ControllerAdvice y devolver una lista de errores de validación:**

```
{
  "error": "Validación fallida",
  "details": [
    "numeroSerie: debe ser mayor que cero",
    "fecha: formato inválido"
  ]
}
```

Respuestas de éxito uniformes

- **Para operaciones de creación, devolver 201 Created con el recurso creado o al menos su identificador en el body.**
- **Para actualizaciones/borrados, devolver 200 OK o 204 No Content según convención. Si se devuelve body, emplear un objeto uniforme, por ejemplo:**

```
{
  "mensaje": "Recurso actualizado correctamente",
  "id": 123
}
```

13.2. Casos comunes y ejemplos de respuestas

1. **Token ausente o mal formado**
    - **Código: 401 Unauthorized**
    - **Body:**

```
{ "error": "Token no proporcionado o inválido." }
```

O bien si se detecta expirado:

```
{ "error": "Token inválido o expirado." }
```

Acceso denegado (recurso de otro usuario o no pertenece)

- **Código: 403 Forbidden**
- **Body:**

```
{ "error": "Acceso denegado para este usuario." }
```

O mensajes específicos: “No puedes modificar ejercicios de otro usuario.”, “Esta semana no pertenece al usuario.”, etc.

Recurso no encontrado

- **Código: 404 Not Found**
- **Body:**

```
{ "error": "Subbloque no encontrado" }
```

O “Ejercicio no encontrado”, etc.

Datos inválidos en el request

- **Código: 400 Bad Request**
- **Body:**

```
{
  "error": "Validación fallida",
  "details": [
    "nombre: no puede estar vacío",
    "fecha: formato inválido, debe ser YYYY-MM-DD"
  ]
}
```

Para simplificar, si no se usa Bean Validation, se puede devolver:

```
{ "error": "El campo grupoMuscular es obligatorio." }
```

Operación exitosa de creación

- **Código: 201 Created**
- **Body:**

```
{
  "mensaje": "Ejercicio añadido correctamente",
  "ejercicio": { ... DTO ... }
}
```

Operación exitosa de actualización

- **Código: 200 OK**
- **Body:**

```
{
  "mensaje": "Semana actualizada correctamente",
  "semana": { "id": 55, "numeroSemana": 3 }
}
```

O simplemente devolver el DTO actualizado si se prefiere:

```
{ "id": 55, "numeroSemana": 3 }
```

Operación exitosa de eliminación

- **Código: 200 OK o 204 No Content**
- **Body (200):**

```
{ "mensaje": "Ejercicio eliminado correctamente." }
```

14. DTOs usados en requests/responses

14.1. DTOs de Autenticación y Registro

- **LoginRequest**
  - **Propósito: Contener credenciales en el endpoint de login.**
  - **Campos:**
    - **String correo — correo electrónico del usuario o entrenador.**
    - **String contrasena — contraseña en texto plano enviada por el cliente.**
  - **Ejemplo JSON:**

```
{
  "correo": "user@example.com",
  "contrasena": "password123"
}
```

PeticionRegistroUsuario

- **Propósito: Datos para registrar un nuevo usuario.**
- **Campos:**
  - **String nombre — nombre completo o nombre de usuario.**
  - **String correo — correo electrónico único.**
  - **String contrasena — contraseña en texto plano.**
- **Ejemplo JSON:**

```
{
  "nombre": "Juan Pérez",
  "correo": "juan@example.com",
  "contrasena": "miClaveSegura"
}
```

UsuarioDTO

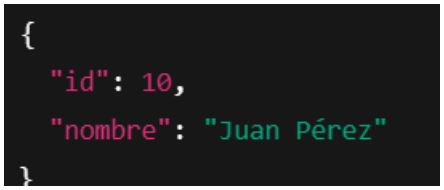
- **Propósito: Representar información de usuario en respuestas (listados, detalles).**
- **Campos:**
  - **Long id — identificador del usuario.**
  - **String nombre — nombre del usuario.**
  - **String correo — correo del usuario.**
  - **String rol — nombre del rol (p.ej., "usuario").**
  - **Long entrenadorId — ID del entrenador asignado, si corresponde (puede ser null).**
- **Ejemplo JSON:**

```
{
  "id": 10,
  "nombre": "Juan Pérez",
  "correo": "juan@example.com",
  "rol": "usuario",
  "entrenadorId": 5
}
```

- **Uso:**
  - **Response de registro (/api/registro).**
  - **Listados de usuarios (GET /api/admin/usuarios).**
  - **Detalles de usuario en EntrenadorDTO.**

UsuarioMiniDTO

- **Propósito: Información mínima de usuario dentro de bloques (para mostrar asignación de bloques).**
- **Campos:**
  - **Long id — ID del usuario.**
  - **String nombre — nombre del usuario.**
- **Ejemplo JSON:**



  - **Uso:**
    - **En BloqueDTO para indicar usuario asignado a un bloque.**

14.2. DTOs de Entrenador

- **EntrenadorDTO**
  - **Propósito: Representar un entrenador con su lista de usuarios.**
  - **Campos:**
    - **Long id — ID del entrenador.**
    - **String nombre — nombre del entrenador.**
    - **String correo — correo del entrenador.**
    - **String rol — nombre del rol (p.ej., "admin").**
    - **Set<UsuarioDTO> usuarios — conjunto de usuarios asignados a este entrenador.**
  - **Ejemplo JSON:**

- **Uso:**
  - **GET /api/admin/entrenadores/{entrenadorId}/usuarios.**

14.3. DTOs de Bloques y Semanas

- **BloqueRequestDTO**
  - **Propósito: Datos para crear o actualizar un bloque.**
  - **Campos:**
    - **String nombre — nombre del bloque.**
    - **Long usuarioId — ID del usuario asignado (opcional al crear, obligatorio al actualizar si se requiere reasignar).**
  - **Ejemplo JSON:**

```
{
  "nombre": "Programa Principiantes",
  "usuarioId": 10
}
```

- **Uso:**
  - **POST /api/admin/bloques.**
  - **PUT /api/admin/bloques/{bloqueId}.**

□ BloqueDTO

- **Propósito: Representar un bloque en respuestas.**
- **Campos:**

- Long id — ID del bloque.
- String nombre — nombre del bloque.
- UsuarioMiniDTO usuario — usuario asignado (o null).
- Long entrenadorId — ID del entrenador propietario.

- **Ejemplo JSON:**

```
{
  "id": 20,
  "nombre": "Programa Avanzado",
  "usuario": { "id": 10, "nombre": "Juan Pérez" },
  "entrenadorId": 5
}
```

- **Uso:**
  - GET /api/admin/bloques.

SemanaRequestDTO

- **Propósito: Datos para crear o actualizar una semana.**
- **Campos:**
  - int numeroSemana — número de la semana dentro del bloque.
- **Ejemplo JSON:**

```
{
  "numeroSemana": 2
}
```

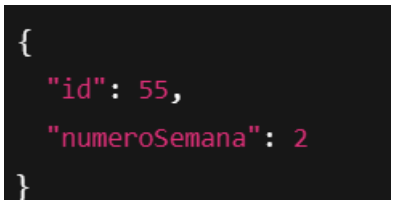
- **Uso:**
  - POST /api/bloques/{bloqueId}/semanas.
  - PUT /api/semanas/{semanaId}.

SemanaDTO

- **Propósito: Representar una semana en respuestas.**
- **Campos:**

- **Long id — ID de la semana.**
- **int numeroSemana — número de la semana.**

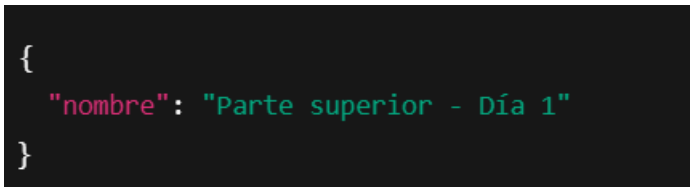
- **Ejemplo JSON:**



- **Uso:**
  - **GET /api/bloques/{bloqueId}/semanas.**
  - **GET /api/semanas/{semanaId}.**

14.4. DTOs de Sub-bloques

- **SubBloqueRequestDTO**
  - **Propósito: Datos para crear o actualizar un sub-bloque.**
  - **Campos:**
    - **String nombre — nombre del sub-bloque.**
  - **Ejemplo JSON:**



- **Uso:**
  - **POST /api/semanas/{semanaId}/subbloques.**
  - **PUT /api/subbloques/{subBloqueId}.**

SubBloqueDTO

- **Propósito: Representar un sub-bloque en respuestas de entrenador.**
- **Campos:**

- **Long id — ID del sub-bloque.**
- **String nombre — nombre del sub-bloque.**

- **Ejemplo JSON:**

```
{
  "id": 101,
  "nombre": "Piernas - Día 2"
}
```

- **Uso:**
  - **GET /api/semanas/{semanaId}/subbloques.**
  - **GET /api/subbloques/{subBloqueId}.**

SubBloqueUsuarioDTO

- **Propósito: Representar un sub-bloque en respuestas de usuario.**
- **Campos:**
  - **Long id — ID del sub-bloque.**
  - **String nombre — nombre del sub-bloque.**
- **Ejemplo JSON:**

```
{
  "id": 101,
  "nombre": "Piernas - Día 2"
}
```

  - **Uso:**
    - **GET /api/usuarios/{usuarioId}/semanas/{semanaId}/subbloques.**

14.5. DTOs de Ejercicios Sugeridos

- **EjercicioSugeridoRequestDTO**
  - **Propósito: Datos para crear o actualizar un ejercicio sugerido por el entrenador.**

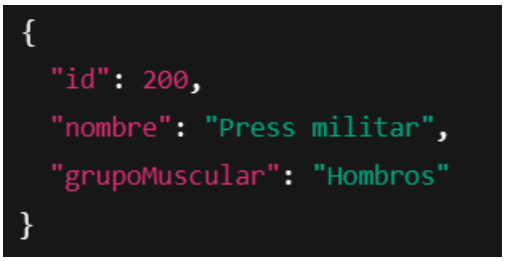
- **Campos:**
  - **String nombre — nombre del ejercicio.**
  - **String grupoMuscular — grupo muscular objetivo.**
  - **Set<SerieSugeridaDTO> series — (opcional) series sugeridas iniciales; puede procesarse en servicio si se desea crear series junto al ejercicio.**
- **Ejemplo JSON:**

```
{
  "nombre": "Press militar",
  "grupoMuscular": "Hombros",
  "series": [
    { "numeroSerie": 1, "repeticiones": 10, "peso": 20.0 },
    { "numeroSerie": 2, "repeticiones": 8, "peso": 22.5 }
  ]
}
```

- **Uso:**
  - **POST /api/subbloques/{subBloqueId}/ejercicios-sugeridos.**
  - **PUT /api/subbloques/ejercicios-sugeridos/{id}.**

EjercicioSugeridoDTO

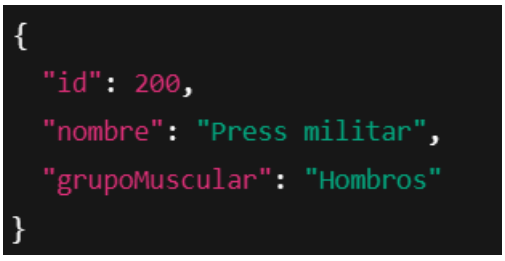
- **Propósito: Representar un ejercicio sugerido en respuestas.**
- **Campos:**
  - **Long id — ID del ejercicio sugerido.**
  - **String nombre — nombre del ejercicio.**
  - **String grupoMuscular — grupo muscular objetivo.**
- **Ejemplo JSON:**



- **Uso:**
  - **GET /api/subbloques/{subBloqueId}/ejercicios-sugeridos.**
  - **GET /api/subbloques/ejercicios-sugeridos/{id}.**
  - **POST y PUT respondiendo con el DTO o con mensaje y ejercicioSugeridoId.**

EjercicioSugeridoUsuarioDTO

- **Propósito: Representar un ejercicio sugerido en vista de usuario (similar a EjercicioSugeridoDTO).**
- **Campos:**
  - **Long id**
  - **String nombre**
  - **String grupoMuscular**
- **Ejemplo JSON:**

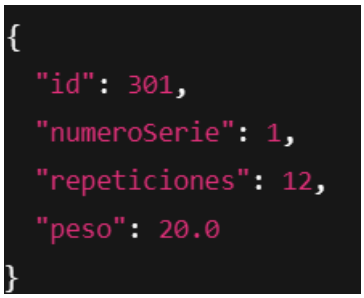


  - **Uso:**
    - **GET /api/usuarios/{usuarioId}/subbloques/{subBloqueId}/ejercicios-sugeridos.**

14.6. DTOs de Series Sugeridas

- **SerieSugeridaDTO**

- **Propósito: Representar o recibir datos de una serie sugerida (p. ej., número de serie, repeticiones, peso).**
- **Campos:**
  - **Long id — ID de la serie sugerida (en responses).**
  - **int numeroSerie — número de serie dentro del ejercicio sugerido.**
  - **Integer repeticiones — repeticiones recomendadas.**
  - **Double peso — peso sugerido.**
- **Ejemplo JSON:**



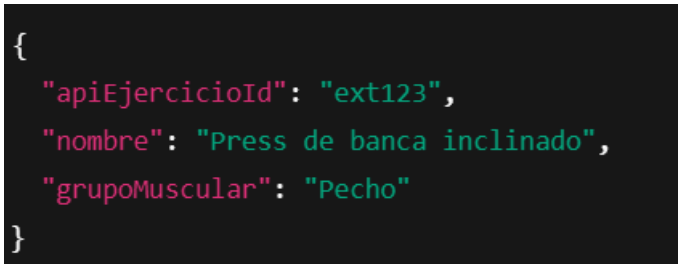
Uso:

- **GET /api/series-sugeridas/ejercicios-sugeridos/{ejercicioId}/series.**
- **POST /api/series-sugeridas/ejercicios-sugeridos/{id}/series (sin id en body, se ignora o se omite).**
- **PUT /api/series-sugeridas/series-sugeridas/{id}.**
- **GET /api/series-sugeridas/usuarios/{usuarioId}/subbloques/{subBloqueId}/ejercicios/{ejercicioId}/series devuelve lista de estos DTOs.**

14.7. DTOs de Ejercicios Personales de Usuario

- **EjercicioUsuarioRequestDTO**
  - **Propósito: Datos para crear o actualizar un ejercicio personal.**
  - **Campos:**

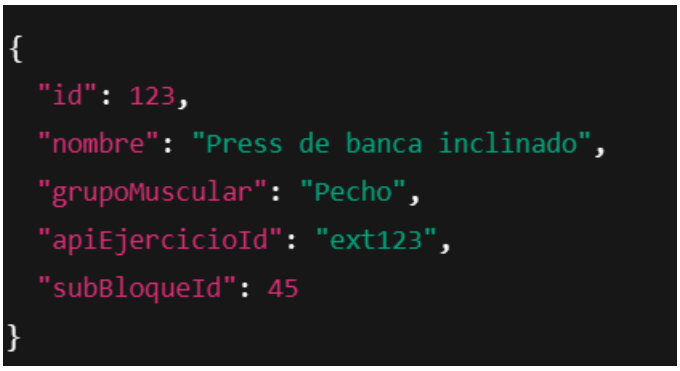
        - **String apiEjercicioId — identificador externo o propio del ejercicio.**
        - **String nombre — nombre descriptivo.**
        - **String grupoMuscular — grupo muscular.**
    - **Ejemplo JSON:**



- **Uso:**
    - **POST /api/usuarios/{usuarioId}/subbloques/{subBloqueId}/ejercicios.**
    - **PUT no existe endpoint para actualizar ejercicio personal, pero se podría añadir.**

EjercicioUsuarioDTO

- **Propósito: Representar un ejercicio personal en respuestas.**
- **Campos:**
    - **Long id — ID en tabla ejercicio_usuario.**
    - **String nombre — nombre descriptivo.**
    - **String grupoMuscular — grupo muscular.**
    - **String apiEjercicioId — identificador externo/propio.**
    - **Long subBloqueId — sub-bloque asociado.**
- **Ejemplo JSON:**



- **Uso:**
  - **GET /api/usuarios/{usuarioId}/subbloques/{subBloqueId}/ejercicios.**
  - **En gráficas: se usa lista de estos DTOs (o al menos su id y apiEjercicioId) para selección.**

14.8. DTOs de Series de Entrenamiento de Usuario

- **SerieEntrenamientoDTO**
  - **Propósito: Representar o recibir datos de una serie de entrenamiento personal.**
  - **Campos:**
    - **Long id — ID de la serie.**
    - **int numeroSerie — número secuencial de la serie.**
    - **Integer repeticiones — repeticiones realizadas.**
    - **Double peso — peso levantado.**
    - **Integer esfuerzoPercibido — escala subjetiva de esfuerzo.**
    - **LocalDate fecha — fecha de la serie (yyyy-MM-dd).**
  - **Ejemplo JSON:**

```
{
  "id": 502,
  "numeroSerie": 1,
  "repeticiones": 10,
  "peso": 50.0,
  "esfuerzoPercibido": 7,
  "fecha": "2025-06-01"
}
```


- **Uso:**
  - **GET /api/usuarios/{usuarioId}/ejercicios/{ejercicioId}/series.**
  - **POST /api/usuarios/{usuarioId}/ejercicios/{ejercicioId}/series.**
  - **PUT /api/usuarios/{usuarioId}/series/{serieId}.**
  - **DELETE no usa body.**

MaxPesoDTO

- **Propósito: Representar el peso máximo por fecha para gráficas.**
- **Campos:**
  - **LocalDate fecha — fecha de entrenamiento.**
  - **Double pesoMaximo — peso máximo levantado ese día.**
- **Ejemplo JSON:**


```
[
  { "fecha": "2025-06-01", "pesoMaximo": 55.0 },
  { "fecha": "2025-06-05", "pesoMaximo": 57.5 }
]
```


  - **Uso:**
    - **GET /api/grafica/{usuarioId}/ejercicios/{ejercicioUsuarioId}/max-pesos.**

14.9. DTOs de Bloques/Semanas/Sub-bloques de Usuario

- **BloqueUsuarioDTO**
    - **Propósito: Representar un bloque asignado al usuario en respuestas de usuario.**
    - **Campos:**
        - **Long id — ID del bloque.**
        - **String nombre — nombre del bloque.**
    - **Ejemplo JSON:**

```
{
  "id": 20,
  "nombre": "Programa Principiantes"
}
```

- **Uso:**
    - **GET /api/usuarios/{usuarioId}/bloques.**

SemanaUsuarioDTO

- **Propósito: Representar una semana dentro de un bloque para el usuario.**
- **Campos:**
    - **Long id — ID de la semana.**
    - **int numeroSemana — número de semana.**
- **Ejemplo JSON:**

```
{
  "id": 55,
  "numeroSemana": 1
}
```

- **Uso:**

- GET /api/usuarios/{usuarioId}/bloques/{bloqueId}/semanas.

SubBloqueUsuarioDTO

- **Propósito: Representar sub-bloque en contexto del usuario.**
- **Campos:**
  - **Long id**
  - **String nombre**
- **Ejemplo JSON:**

```
{
  "id": 101,
  "nombre": "Piernas - Día 2"
}
```

15. Seguridad y JwtFilter

La seguridad de la API se basa en JWT (JSON Web Tokens), con un filtro (JwtFilter) que intercepta cada petición para validar el token y establecer la autenticación en el contexto de Spring Security. A continuación se explica el flujo, la configuración y recomendaciones.

15.1. Flujo de autenticación

1. **Login**
   - **Endpoint: POST /api/auth/login**
   - **El cliente envía LoginRequest (correo y contrasena).**
   - **El controlador (AuthController) busca en UsuarioRepository o EntrenadorRepository la entidad con ese correo.**
   - **Verifica la contraseña con PasswordEncryptor.checkPassword(raw, hashed).**

- **Si coincide, genera un JWT con jwtUtil.create(email, rol), donde se incluye en el claim "password" el rol o contraseña (según implementación actual).**
    - **Nota: Incluir la contraseña en el token no es una práctica recomendada. Lo habitual es incluir el usuario (sub) y, opcionalmente, rol en claims, sin exponer la contraseña.**
- **Devuelve JSON con { "token": "<jwt>", "id": <id>, "correo": ..., "nombre": ..., "roles": ["rol"] }.**

2. **Envio de peticiones autenticadas**
    - **El cliente incluye en cada petición protegida el header:**

```
Authorization: Bearer <token>
```

1.
    - **El token se extrae en el filtro JwtFilter.**
2. **JwtFilter (OncePerRequestFilter)**
    - **Se ejecuta antes de UsernamePasswordAuthenticationFilter.**
    - **En doFilterInternal:**
        1. **Obtener request.getRequestURI(). Si la ruta empieza con /api/auth/ (login) o /api/registro, se omite validación y se continúa el filter chain.**
        2. **Obtener header Authorization. Si ausente o no empieza con "Bearer ", se deja pasar sin autenticación (requests posteriores pueden fallar por falta de autenticación).**
        3. **Extraer token substring tras "Bearer ".**
        4. **Intentar extraer email y password (en este caso, “rol” o contraseña) con jwtUtil.extractEmail(token) y jwtUtil.extractPassword(token). Si falla (token mal formado o expirado) → devuelve 401 Unauthorized y**

mensaje “Error: Token inválido o expirado”, y no continua.

5. **Si email extraído y SecurityContextHolder.getContext().getAuthentication() es null:**
    - **Buscar en UsuarioRepository.findByCorreo(email). Si existe, construir UserDetails con username=email, password=usuario.getContrasena(), roles desde usuario.getRol().getNombre().toUpperCase().**
    - **Si no, buscar en EntrenadorRepository.findByCorreo(email). Si existe, construir UserDetails con roles “ADMIN”.**
6. **Verificar jwtUtil.validateToken(token, email, password):**
    - **En la implementación actual, compara el claim extraído de token (extractPassword) con el password o rol esperado.**
    - **Mejora recomendada: En lugar de incluir contraseña en el claim, incluir solo el email (subject) y el rol en un claim separado (role), y en validateToken verificar firma y expiración. No extraer la contraseña ni compararla.**
7. **Si válido, crear UsernamePasswordAuthenticationToken con userDetails y setear en SecurityContextHolder.**
8. **Continuar el filter chain.**

3. **SecurityConfig**
    - **Configura SecurityFilterChain en un bean:**

        - **cors() habilitado con orígenes permitidos (p.ej. http://localhost:8100).**
        - **csrf().disable().**
        - **authorizeHttpRequests define rutas públicas (/actuator/**, /api/auth/**, /api/registro, y en algunos casos /api/admin/.../asignar-entrenador-auto/**), permitiendo sin token.**
        - **Rutas /api/admin/** requieren rol ADMIN.**
        - **Rutas /api/usuarios/** requieren rol USUARIO o ADMIN.**
        - **Otras rutas requieren autenticación.**
        - **sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS) (sin sesiones HTTP).**
        - **addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class).**
    - **Bean AuthenticationManager para que Spring Security pueda validar credenciales en login (aunque en AuthController no se usa directamente, se hace la validación manual con PasswordEncryptor).**
    - **Bean PasswordEncoder de tipo BCryptPasswordEncoder para encriptar contraseñas en registro y compararlas en login.**
4. **CustomUserDetailsService**
    - **Implementa UserDetailsService.loadUserByUsername(correo) usado por Spring Security cuando se autentica con credenciales (p.ej., en flujos donde se invoca authenticationManager.authenticate(...)).**

- Busca en UsuarioRepository.findByCorreoWithRol(correo), si lo encuentra, construye UserDetails con roles según usuario.getRol().getNombre().toUpperCase().
- Si no, busca en EntrenadorRepository.findByCorreo(correo), construye UserDetails con rol “ADMIN”.
- Si no existe, lanza UsernameNotFoundException.

5. Validación de rutas
    - En cada controlador, se extrae token y se valida con jwtUtil.validateToken(token) o con usuarioService.obtenerUsuarioDesdeToken(token) / entrenadorService.obtenerEntrenadorDesdeToken(token).
    - Se verifica la pertenencia de recursos (ID coincide o es cliente del entrenador).
    - Los endpoints para admins/entrenadores incluyen cheques adicionales de rol:
        - E.g., en BloqueController, EntrenadorController, se extrae el correo del token, busca entrenadorRepository.findByCorreo(correo), y si no existe → 403 Forbidden.
        - Verificar entrenador.getRol().getNombre().equalsIgnoreCase("admin") para ciertas operaciones estrictamente admin.

15.2. Elementos clave en JwtFilter y JWTUtil

- JWTUtil
    - Genera tokens con firma HMAC-SHA256 (Keys.secretKeyFor(SignatureAlgorithm.HS256) genera clave en runtime).
    - create(String email, String password):
        - Incluye setSubject(email).

        - **Claim "password" con el valor pasado (actualmente se usa para validar rol o contraseña).**
        - **Expiración de 7 días.**
        - **Firma con signingKey.**
    - **Métodos de extracción:**
        - **extractEmail(token) obtiene subject.**
        - **extractPassword(token) obtiene claim "password".**
    - **Validación:**
        - **validateToken(token) verifica expiración.**
        - **validateToken(token, email, password) compara extraídos con parámetros y revisa expiración.**
    - **Mejora recomendada:**
        - **Incluir en claims solo información necesaria (subject=email, claim “role”:rol). No incluir contraseña.**
        - **En validateToken, verificar firma y expiración únicamente; para autorización se obtiene rol desde base de datos o desde claim seguro.**
        - **Usar clave fija o inyectada via configuración (properties), no generada en runtime cada vez que arranca la app, de lo contrario los tokens antiguos quedarían inválidos tras reinicio.**
- **JwtFilter**
    - **Omite rutas públicas.**
    - **Extrae token, valida token, construye UserDetails y setea autenticación.**
    - **Imprime logs de debugging (System.out.println). En producción, usar logger con niveles adecuados.**

15.3. SecurityConfig

- **CORS:**
  - **Orígenes permitidos: p.ej. http://localhost:8100. No usar "*".**
  - **Métodos permitidos: GET, POST, PUT, DELETE, OPTIONS.**
  - **Headers permitidos: *.**
  - **allowCredentials(true) para enviar cookies si fuera necesario (aunque el esquema es stateless con JWT).**
- **CSRF:**
  - **Deshabilitado (csrf.disable()) porque la API es stateless y se usa token en headers.**
- **Rutas públicas vs protegidas:**
  - **/api/auth/**, /api/registro sin autenticación.**
  - **Rutas específicas permitidas sin token (por ejemplo, asignación automática de entrenador tras registro).**
  - **/api/admin/** restringido a rol ADMIN.**
  - **/api/usuarios/** restringido a rol USUARIO o ADMIN.**
  - **Otras rutas requieren autenticación.**
- **SessionManagement:**
  - **SessionCreationPolicy.STATELESS: no se usan sesiones HTTP; toda la info de sesión via JWT.**
- **Inyección de JwtFilter:**
  - **addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class) para que JwtFilter procese antes de la autenticación por formulario.**
- **AuthenticationManager:**

- **Permite autenticar credenciales en login si se usara authenticationManager.authenticate(...), aunque en la implementación actual se realiza la validación manual.**

16. Notas de despliegue del backend

16.1. Requisitos Previos

1. **Repositorio y código fuente**
   - **El código del backend (Spring Boot) debe estar actualizado en la máquina de desarrollo o en el repositorio git local/clonado.**
   - **Asegurarse de que la configuración (application.properties o application.yml) para entornos de despliegue (variables de entorno, endpoints de BD, credenciales seguras) esté correctamente preparada. No se incluyen valores sensibles en el repositorio; se usan variables de entorno o vault según la política de la organización.**
2. **Herramientas locales de desarrollo**
   - **Java JDK: versión compatible con Spring Boot (por ejemplo, Java 17+ si el proyecto está configurado así).**
   - **Maven Wrapper (mvnw): ya presente en el proyecto. Permite compilar el JAR sin depender de una instalación global de Maven.**
   - **Docker y Docker Compose: instalados en la máquina/servidor de despliegue. El servidor debe tener Docker y Docker Compose instalados y con permisos para arrancar contenedores.**
3. **Acceso a servidor remoto**
   - **Disponer de las credenciales SSH (usuario, clave o clave pública autorizada) tal como se detalla en "Datos de conexión…".**

- **Acceso SCP/SFTP para copiar artefactos al servidor, o bien pipeline automatizado (CI/CD) que tome el JAR y lo suba automáticamente.**

4. **Estructura de despliegue en servidor**
    - **Directorio base de despliegue, por ejemplo ~/fittrack-deploy/backend/.**
    - **Dentro, disponer de:**
        - **Un Docker Compose file (docker-compose.yml) que defina el servicio del backend (por ejemplo, imagen o contexto local), su red, variables de entorno, volúmenes si aplica.**
        - **Un directorio target/ o similar donde se colocan los artefactos (JAR) antes de arrancar.**
    - **El servidor debe tener acceso a la base de datos (MySQL/PostgreSQL u otra) en entorno productivo o staging; en el Compose/variables se apuntará al host/puerto/credenciales adecuados.**
5. **Variables de entorno y secretos**
    - **No incluir credenciales en texto plano. Usar variables de entorno en Docker Compose o mecanismos de inyección de secretos (Docker secrets, Kubernetes Secrets, Vault).**
    - **Revisar que en application.properties o configuración de Spring Boot se referencien variables de entorno (p. ej. ${DB_URL}, ${DB_USER}, ${DB_PASSWORD}, ${JWT_SECRET} si aplica).**
6. **Versionado de artefactos**
    - **Mantener el número de versión en pom.xml o build.gradle acorde a semver o política interna.**
    - **Cada despliegue usa el JAR con nombre que incluya la versión, p.ej. FitTrack-0.0.1-SNAPSHOT.jar.**

- Opcional: renombrar o archivar versiones anteriores en el servidor para rollback si fuera necesario.

16.2. Pasos de compilación y empaquetado (máquina de desarrollo o CI/CD)

Partimos de tu ejemplo de comandos en Windows PowerShell / CMD:

1. **Ir al directorio del proyecto**

```
cd /d E:\Proyecto\Back_Conjunto\Back\FitTrack
```

- **Asegurarse de estar en la raíz donde está el wrapper de Maven (mvnw.cmd o mvnw).**

Compilar y empaquetar el JAR

```
.\mvnw.cmd clean package -DskipTests
```

- **clean package: limpia compiles previas y genera un JAR en target/.**
- **-DskipTests: omite la ejecución de tests unitarios/integración. En producción, conviene asegurarse de que las pruebas pasan en CI, pero en despliegue rápido se puede omitir localmente si ya se verificó en CI.**

Verificar la generación del JAR

```
dir .\target\FitTrack-0.0.1-SNAPSHOT.jar
```

- **Confirma que el archivo existe y está actualizado (timestamp).**
- **Si la versión cambia, comprobar nombre: puede ser FitTrack-<versión>.jar.**

Preparar artefacto para copia

- **Opcional: renombrar o mover el JAR a ubicación temporal de despliegue local, por ejemplo:**

copy ".\target\FitTrack-0.0.1-SNAPSHOT.jar" "E:\fittrack-deploy\backend\target\FitTrack-0.0.1-SNAPSHOT.jar"

1.

   - Este paso organiza una carpeta local (E:\fittrack-deploy\backend\target\) que luego se sincroniza al servidor.

2. (Opcional) Versionado / Backup local

   - Mantener copias de versiones previas si es necesario rollback.
   - Anotar la versión exacta desplegada (p.ej. en un changelog o ticket).

16.3. Transferencia al servidor

1. Copiar el artefacto al servidor

   - Usando SCP (o SFTP) basado en la guía de conexión:

   scp "E:\fittrack-deploy\backend\target\FitTrack-0.0.1-SNAPSHOT.jar" sergiolg25@192.168.7.157:~/fittrack-deploy/backend/target/

- Aquí:

   - sergiolg25@192.168.7.157: usuario e IP del servidor; los detalles (puerto SSH, clave privada, ruta) están en "Datos de conexión…".
   - Ruta destino en servidor: ~/fittrack-deploy/backend/target/.

- Consejo: si se usa CI/CD, reemplazar este paso manual por pipeline que haga upload seguro, o usar rsync para sincronizar sólo cambios.

Conexión SSH al servidor

```
ssh sergiolg25@192.168.7.157
```

Autenticarse con clave o contraseña según política.

Navegar a directorio base de despliegue:

```
cd ~/fittrack-deploy
```

1.

- **Revisar que permisos de usuario permitan ejecutar Docker Compose.**

16.4. Despliegue con Docker Compose

Se asume que en ~/fittrack-deploy existe un archivo docker-compose.yml que define el servicio del backend. Ejemplo genérico de sección en docker-compose.yml:

```yaml
version: '3.8'

services:
  backend:
    image: fittrack-backend:${TAG}     # si se construye o se etiqueta; alternativamente build:
    build:
      context: ./backend
      dockerfile: Dockerfile
    volumes:
      - ./backend/target/FitTrack-0.0.1-SNAPSHOT.jar:/app/FitTrack.jar:ro
    environment:
      - SPRING_PROFILES_ACTIVE=prod
      - DB_URL=jdbc:mysql://db-host:3306/fittrack
      - DB_USER=...
      - DB_PASSWORD=...
      # Otras vars, p.ej. JWT secret, configuración de logging, etc.
    ports:
      - "8080:8080"
    depends_on:
      - database  # si se define en el mismo Compose u externa; en caso externo, no es necesario
    restart: always

  # ... otros servicios como base de datos, si se administran aquí
```

- **Contexto: puedes elegir montar el JAR directamente o bien usar un Dockerfile que copie el JAR al contenedor y ejecute java -jar FitTrack.jar.**
- **Variables de entorno: deben inyectarse de forma segura (usar un archivo .env que no se suba a repo, o Docker secrets).**
- **Perfiles de Spring: usar perfil prod que cargue configuración adecuada.**

3. **Reconstruir y levantar containers**
   Desde ~/fittrack-deploy ejecutar:

```
docker-compose up -d --build
```

--build fuerza la reconstrucción de la imagen si el Dockerfile está en el repo o si se monta el JAR, servirá para recrear el contenedor con la nueva versión del JAR.

-d levanta en segundo plano.

Si no se usa Dockerfile (solo se monta el JAR en volumen), puede bastar con:

```
docker-compose up -d
```

- **puesto que el servicio detectará el nuevo JAR montado y reiniciará el contenedor (dependiendo de la configuración de reinicio).**

Verificar containers

```
docker ps
```

Confirmar que el contenedor del backend esté "Up" y sin reinicios constantes.

Si hay problemas, consultar logs:

```
docker-compose logs backend
# o
docker logs <container_id>
```

Chequeo de salud

- **Probar endpoint de salud o actuator (si está habilitado), p.ej. GET http://<host>:8080/actuator/health para comprobar que la aplicación arranca correctamente y conecta a la base de datos.**
- **Realizar pruebas rápidas: invocar un endpoint autenticado con token de prueba para verificar funcionamiento.**

Rollback

- **Si la nueva versión falla de forma crítica, detener el contenedor:**

```
docker-compose down
```

17. Diagrama ER de la base de datos