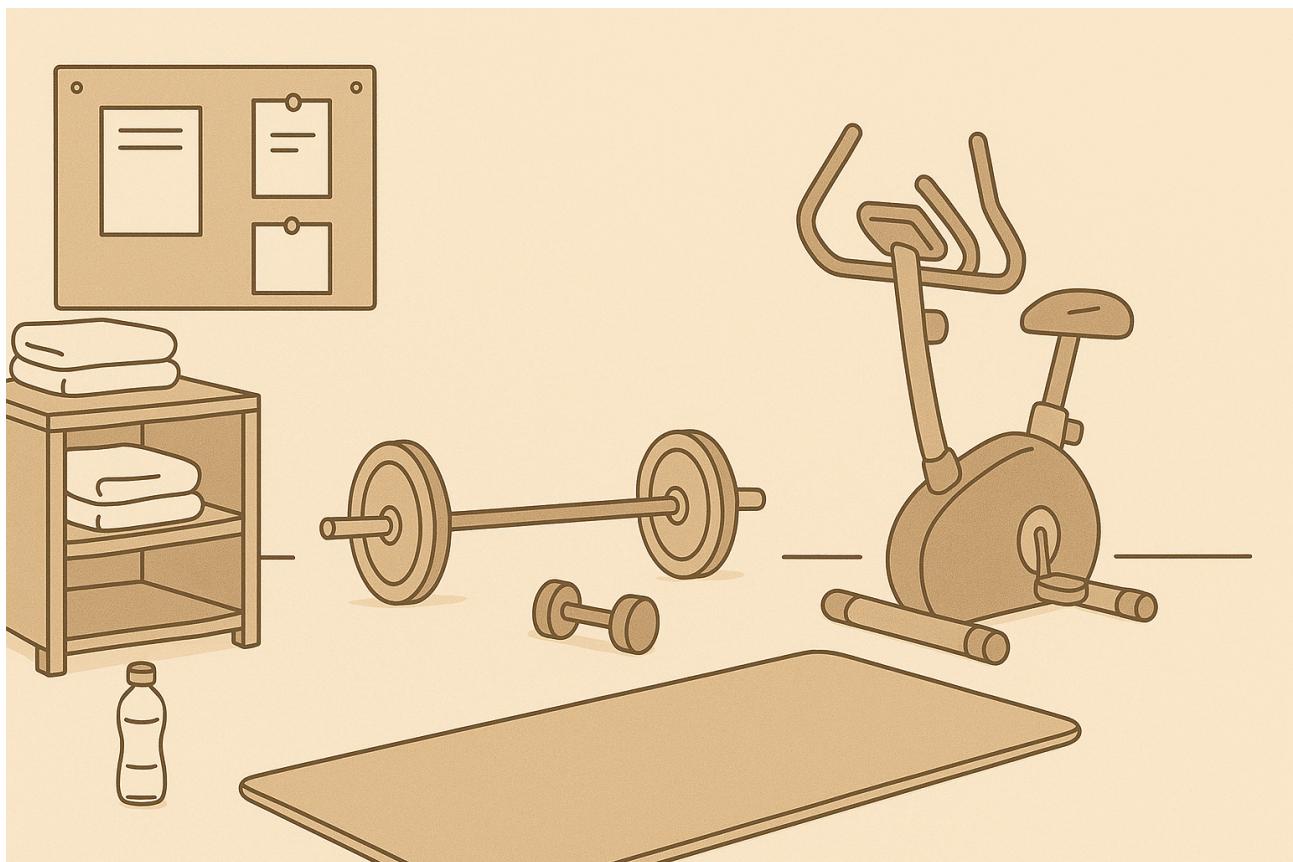


**DOCUMENTACIÓN RUTIFY** Plataforma deportiva



TFG de Nicolás De Gomar (2º DAM, IES Rafael Alberti 2024/25)

# Índice:

<b>1. Introducción.....</b>	<b>4</b>
1.1 Expectativas y Objetivos.....	4
1.2 Antecedentes.....	4
1.3 Desarrollo de la Idea.....	5
1.4 Aplicaciones Similares.....	6
<b>2. Descripción.....</b>	<b>7</b>
<b>3. Instalación.....</b>	<b>8</b>
<b>4. Prototipado.....</b>	<b>9</b>
4.1 pantalla inicio sesion.....	9
4.2 pantalla registro.....	10
4.3 pantalla búsqueda rutinas.....	11
4.4 pantalla detalles de rutina.....	12
4.5 pantalla crear rutina.....	13
4.6 pantalla realizar ejercicio.....	14
4.7 Pantalla usuario Mi zona.....	15
4.8 reto diario.....	16
4.9 elegir ejercicios.....	17
<b>5. Diseño Funcional.....</b>	<b>19</b>
5.1 Entidad Relación.....	19
5.2 UsuariosController.....	20
5.2.1 /v1/usuarios.....	24
5.2.2 usuario se registra.....	25
5.2.3 usuario inicia sesión.....	26
5.2.4 usuario elimina su cuenta.....	26
5.2.5 usuario busca a más usuarios.....	28
5.2.6 usuario obtiene detalles de usuario.....	29
5.2.7 usuario actualiza su cuenta.....	30
5.2.8 usuario verifica si es admin.....	31
5.2.9 usuario reto diario.....	31
5.2.10 usuario cambiar indumentaria.....	32
5.3 RutinaController.....	33
5.3.1 /v1/rutinas.....	35
5.3.2 usuario crea una rutina.....	36
5.3.3 usuario crea ver rutinas.....	38
5.3.4 usuario crea ver rutinas.....	39
5.3.5 usuario obtener detalles rutina.....	40
5.3.6 usuario obtener rutinas por autor.....	41
5.3.7 usuario elimina una rutina.....	42
5.4 VotosController.....	43
5.4.1 /v1/rutinas.....	44
5.4.2 usuario registra voto.....	45
5.4.3 usuario obtiene votos.....	46

5.4.4 usuario actualiza su voto.....	47
5.4.5 usuario se registra.....	48
5.4.6 usuario obtiene votos por autor.....	49
5.5 StripeController.....	50
5.5.1 /v1/pagos.....	50
5.5.1 usuario crea intención de pago con Stripe.....	51
5.5.2 Stripe webhook — manejo de notificaciones de pago.....	52
5.6 ReportesController.....	53
5.6.1 /v1/reportes.....	54
5.6.2 Reportar usuario.....	54
5.7 ModeracionController.....	55
5.7.1 /v1/moderacion.....	55
5.7.2 Verificar comentarios pendientes de moderación.....	56
5.7.3 Eliminar comentario por administrador.....	57
5.7.4 Obtener usuarios reportados.....	58
5.7.5 Eliminar usuario reportado.....	59
5.8 EstadisticasDiariasController.....	60
5.8.1 /v1/estadisticasDiarias.....	61
5.8.2 Obtener últimos 5 pesos registrados.....	62
5.8.3 Actualizar estadísticas diarias.....	62
5.8.4 Obtener estadísticas diarias de un día.....	63
5.9 CoinPackController.....	64
5.9.1 /v1/coin-packs.....	65
5.9.2 usuario se registra.....	65
5.10 CosmeticoController.....	66
5.10.1 /v1/rutinas.....	67
5.10.2 Obtener lista de cosméticos disponibles.....	67
5.11 ComprasController.....	68
5.11.1 /v1/compras.....	69
5.11.2 Obtener cosméticos comprados por el usuario.....	69
5.11.3 Registrar una compra de cosmético.....	70
5.12 EjerciciosController.....	71
5.12.1 /v1/ejercicios.....	72
5.12.2 Crear un ejercicio personalizado.....	73
5.12.3 Obtener reto diario.....	74
5.12.4 Obtener lista de ejercicios.....	75
5.13 ComentariosController.....	76
5.13.1 /v1/comentarios.....	78
5.13.3 Crear un comentario.....	79
5.13.4 Obtener comentarios principales públicos.....	80
5.13.5 Obtener comentario principal y sus respuestas.....	81
5.13.6 Crear respuesta a un comentario.....	82
5.13.7 Eliminar comentario o respuesta.....	83
5.13.8 Aprobar comentario.....	84

5.13.9 Obtener comentarios por autor.....	85
5.13.10 Obtener comentarios por nombre de autor.....	86
<b>5.14 EstadisticasController.....</b>	<b>88</b>
5.14.1 /v1/rutinas.....	89
5.14.2 Obtener estadísticas de usuario.....	90
5.14.3 Crear estadísticas de usuario.....	91
5.14.4 Actualizar estadísticas de usuario.....	92
<b>5.15 Cliente móvil.....</b>	<b>93</b>
5.15.1 Grafo de navegación.....	93
5.15.1 Arquitectura general.....	93
5.15.2 Flujo Registro.....	94
5.15.3 Flujo Login.....	95
5.15.4 Flujo Rutinas.....	96
5.15.5 Flujo Detalles Rutina.....	97
5.15.6 Búsqueda online de rutinas.....	99
5.15.7 Flujo actividad del usuario.....	100
5.15.8 Flujo Crear rutina personalizada.....	101
5.15.9 Flujo realizar rutina.....	102
5.15.10 Pantalla del perfil del usuario.....	104
5.15.11 Pantalla estadísticas diarias del usuario.....	105
5.15.12 Pantalla perfil.....	107
5.15.13 Pantalla foro.....	108
5.15.14 Pantalla detallesComentario.....	109
5.15.15 Pantalla Buscador.....	110
5.15.15 Pantalla Ajustes.....	111
<b>6. Desafíos y problemas afrontados.....</b>	<b>112</b>
<b>7. Desarrollo.....</b>	<b>113</b>
<b>8. Pruebas.....</b>	<b>114</b>
<b>9. Distribución.....</b>	<b>115</b>
<b>10. Manual.....</b>	<b>116</b>
<b>11. Conclusiones.....</b>	<b>122</b>
<b>12. Índice de tablas e imágenes.....</b>	<b>123</b>
<b>13. Bibliografía.....</b>	<b>127</b>
11.1 maquetación:.....	127
11.2 Mongodb:.....	127

# 1. Introducción

## 1.1 Expectativas y Objetivos

En un contexto donde la actividad física cobra cada vez mayor relevancia para la salud y el bienestar, Rutify surge con la intención de brindar una solución accesible, intuitiva y motivadora, especialmente para quienes desean escapar de las rutinas tradicionales y repetitivas. El objetivo principal del proyecto es ofrecer rutinas dinámicas y variadas, generadas por la comunidad, que se adapten a distintos niveles y objetivos sin caer en la clásica estructura monótona de “lunes pecho, martes espalda...”. Además, Rutify incorporará un espacio para cursos teóricos presentados de forma clara y sencilla, permitiendo que los creadores de contenido puedan compartir sus conocimientos y recibir una compensación económica justa por su labor formativa.

## 1.2 Antecedentes

A lo largo de los últimos años, el auge de las aplicaciones móviles de fitness ha transformado la manera en que las personas se entrena. Sin embargo, tras probar varias de las opciones más populares disponibles en Google Play —como Ejercicios en casa: Sin equipo, Muscle Booster o Fitify— es evidente que muchas de estas soluciones presentan limitaciones importantes. Por ejemplo, algunas apps siguen un plan de evolución lineal, obligando al usuario a empezar desde un punto fijo con rutinas que pueden no ser de su agrado, como ocurre en Ejercicios en casa, que además promete resultados en plazos poco realistas para una sociedad que suele carecer de constancia. Otras, como Muscle Booster, adoptan un enfoque mecanicista, tratando al usuario como una máquina, sin posibilidad de personalización real ni interacción con otros usuarios. Y en el caso de Fitify, aunque se promueven entrenamientos divididos por días, estos suelen estar prefabricados y carentes de flexibilidad, repitiendo esquemas tradicionales y monótonos.

Además, la mayoría de estas aplicaciones carecen por completo de funciones comunitarias, lo que impide comparar progresos, compartir resultados o aprender de otros. Esta falta de dinamismo, personalización y conexión social representa una gran oportunidad para Rutify, que apuesta por un enfoque mucho más flexible, comunitario y motivador.

### **1.3 Desarrollo de la Idea**

La idea de Rutify nace con la intención de romper la monotonía de los planes de entrenamiento tradicionales, donde los días ya vienen predefinidos y las opciones del usuario son limitadas. En lugar de imponer una rutina cerrada, Rutify apuesta por la libertad y personalización, permitiendo que cada persona entrene el grupo muscular que desee según sus propios objetivos y preferencias.

El núcleo de la aplicación es un buscador de rutinas creadas por la comunidad, que ofrece una gran variedad de entrenamientos dinámicos, clasificados por nivel, por grupo muscular, por el tipo de equipo necesario, e incluso por el entorno (casa o gimnasio). Estas rutinas incluyen descripciones detalladas, imágenes explicativas y valoraciones de otros usuarios, lo que garantiza una guía clara y confiable.

Además, cada sesión completada se registra, contribuyendo al progreso del usuario tanto a nivel estadístico como visual, mediante un avatar evolutivo que refleja el esfuerzo invertido. Esta combinación entre libertad, gamificación y contenido generado por usuarios convierte a Rutify en una herramienta motivadora y flexible, adaptada a los distintos caminos que cada persona puede seguir en su desarrollo físico.

Como parte del enfoque comunitario de la app, Rutify incorpora un sistema de comentarios que fomenta la interacción entre usuarios. Cualquier persona puede publicar un comentario, ya sea para compartir dudas, recomendaciones o experiencias sobre una rutina. A su vez, estos comentarios permiten respuestas, creando hilos de conversación útiles para resolver preguntas comunes, aclarar ejercicios o simplemente motivarse mutuamente. Por ejemplo, un usuario puede subir una imagen preguntando cómo se realiza un ejercicio, y otro responder con una explicación técnica o consejos prácticos. Esta funcionalidad convierte a Rutify en un espacio colaborativo, donde el conocimiento se construye colectivamente.

Para mantener un entorno seguro y de calidad, Rutify también implementa un sistema de reportes, que permite a los usuarios denunciar contenido inapropiado o comportamientos ofensivos. Cada reporte incrementa un contador asociado al usuario reportado, lo que permite a los administradores tomar medidas si se detecta un patrón de conducta indebida. Además, todas las imágenes subidas por los usuarios pasan por un proceso de verificación manual/automática, que filtra contenido no relacionado o inapropiado antes de que se muestre públicamente. De esta forma, Rutify garantiza una experiencia positiva, responsable y enfocada al bienestar de sus usuarios.

## **1.4 Aplicaciones Similares**

Existen múltiples aplicaciones en el mercado como Freeletics, Fitbod o Nike Training Club que ofrecen planes de entrenamiento, pero suelen enfocarse en usuarios con conocimientos previos o requieren altos niveles de personalización desde el inicio, lo que puede resultar abrumador para principiantes. Por otro lado, apps como Fitify o Seven proponen entrenamientos simples y accesibles, pero carecen de una narrativa atractiva o de un sistema de progresión visual que mantenga la motivación a largo plazo.

Rutify se diferencia al combinar la estructura guiada con una capa lúdica basada en la evolución de un avatar, simplificando la experiencia sin renunciar a la personalización. Pero lo que realmente la distingue es su fuerte enfoque comunitario y su sistema abierto de rutinas: los usuarios pueden buscar, valorar, comentar y responder sobre entrenamientos creados por otros miembros de la comunidad. Esta interacción no solo enriquece el contenido, sino que genera un entorno de apoyo entre usuarios de distintos niveles, facilitando el aprendizaje y la constancia.

A futuro, Rutify contempla la integración de gimnasios locales, permitiendo que estos puedan ofrecer directamente sus propios planes de entrenamiento dentro de la app, fortaleciendo aún más la conexión entre el mundo digital y el entrenamiento presencial.

## 2. Descripción

Rutify es una aplicación de entrenamiento diseñada para **personas de 16 a 60 años**, pensada tanto para quienes están dando sus **primeros pasos en el mundo del fitness** como para **usuarios con experiencia** que buscan **variedad y motivación**. Su interfaz intuitiva y modular se compone de secciones como el buscador de rutinas, perfil de usuario, estadísticas de progreso y un módulo de comunidad donde los usuarios pueden compartir dudas, consejos e ideas.

La app permite buscar rutinas personalizadas según objetivo, nivel de dificultad o equipo disponible (con o sin material, en casa o en gimnasio), y registrar los entrenamientos completados para visualizar el progreso. Cada rutina está acompañada de una descripción detallada, recursos visuales y una valoración comunitaria basada en la experiencia de otros usuarios.

Para mejorar la accesibilidad, Rutify incluye funciones como modo oscuro para entrenar cómodamente en cualquier momento del día y la posibilidad de aumentar el tamaño del texto, adaptándose así a distintos rangos de edad y necesidades visuales.

La experiencia se gamifica mediante un avatar evolutivo, que refleja visualmente el esfuerzo del usuario a lo largo del tiempo. Además, cualquier miembro de la comunidad puede aprender a través de los comentarios, compartir conocimientos con otros y hasta publicar sus propias rutinas, fomentando un entorno colaborativo.

Rutify distingue entre usuarios gratuitos, usuarios premium y gimnasios (estos últimos en futuras actualizaciones), ofreciendo funcionalidades diferenciadas según el tipo de cuenta. Toda la información está respaldada por una base de datos en la nube con autenticación segura mediante Firebase, garantizando tanto la privacidad como la disponibilidad de los datos en todo momento.

### **3. Instalación**

Actualmente, Rutify se encuentra en fase de distribución inicial y puede instalarse de forma manual mediante un archivo APK. A continuación, se describen los pasos para realizar la instalación en dispositivos Android, así como las previsiones para su publicación en Google Play:

#### **Instalación mediante APK (versión actual)**

<https://www.dropbox.com/scl/fi/2b0wcrrw9pb3xtd587uza/app-debug.apk?rlkey=eommkbbmvot9gcp02vfq rng7m&st=suj7d86m&dl=0>

En tu dispositivo Android:

Ve a Ajustes > Seguridad (o Privacidad, según el modelo).

Activa la opción "Permitir instalación de aplicaciones de fuentes desconocidas" si aún no está habilitada. Abre el archivo APK descargado desde tu gestor de archivos o directamente desde las notificaciones.

Sigue las instrucciones del instalador y acepta los permisos necesarios para completar la instalación.

Una vez instalada, podrás abrir Rutify desde el menú de aplicaciones como cualquier otra app.

#### **Instalación futura desde Google Play**

En futuras versiones, Rutify estará disponible en Google Play, lo que facilitará su instalación y actualización automática. Cuando esté disponible, el proceso será tan sencillo como:

Acceder a la Google Play Store.

Buscar "Rutify".

Pulsar en "Instalar".

Abrir la app directamente desde Google Play o desde el menú de aplicaciones.

## 4. Prototipado

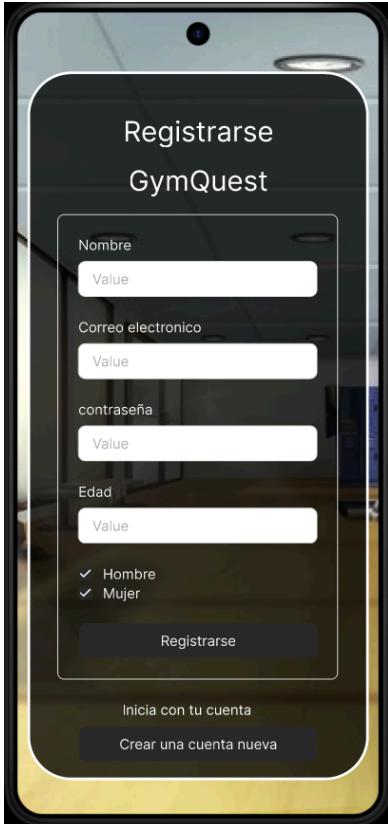
### 4.1 pantalla inicio sesion



Una pantalla sencilla donde el usuario se registra con el correo que previamente introdujo en la pantalla de registro y su contraseña en caso de que falle las credenciales o no exista aparecerá un snack bar advirtiendo del fallo que está cometiendo

Imagen: Pantalla de inicio sesión.

#### 4.2 pantalla registro



The image shows a smartphone displaying the 'Registrarse' (Register) screen for the 'GymQuest' app. The screen has a dark background with white text and input fields. It includes fields for 'Nombre' (Name), 'Correo electronico' (Email), 'contraseña' (Password), and 'Edad' (Age). Below these are two checkboxes: 'Hombre' (Male) and 'Mujer' (Female), both of which are checked. At the bottom of the form is a 'Registrarse' (Register) button. Below the form, there are two links: 'Inicia con tu cuenta' (Log in with your account) and 'Crear una cuenta nueva' (Create a new account).

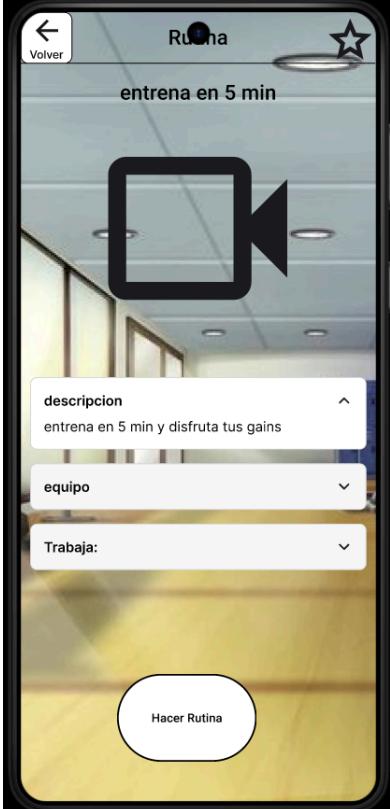
Pantalla de registro donde el usuario se registrara informandonos a la plataforma de su sexo y su edad, para poder mostrarles las rutinas mas acertadas para su caso, segun nuestro futuro algoritmo

Imagen: Pantalla de registro.

#### 4.3 pantalla búsqueda rutinas

	<p>Pantalla de búsqueda de rutinas donde le saldrán al usuario rutinas a realizar y él mismo escoge cual le apetece con total libertad, las estrellas significan si están rellenas es que es una rutina de pago, las cuales las rutina de pagos se implementan a futuro</p>
Imagen: Pantalla de búsqueda rutinas.	

#### 4.4 pantalla detalles de rutina



Pantalla donde te muestra, el equipo necesario para completar esa rutina, el creador puede compartir una descripción personal con consejos útiles, explicando por qué esta rutina es efectiva, en qué se enfoca (como fuerza, resistencia, tonificación, etc.) y para qué tipo de usuario está pensada y que tipo de grupo muscular trabaja

Imagen: Pantalla detalles rutinas.

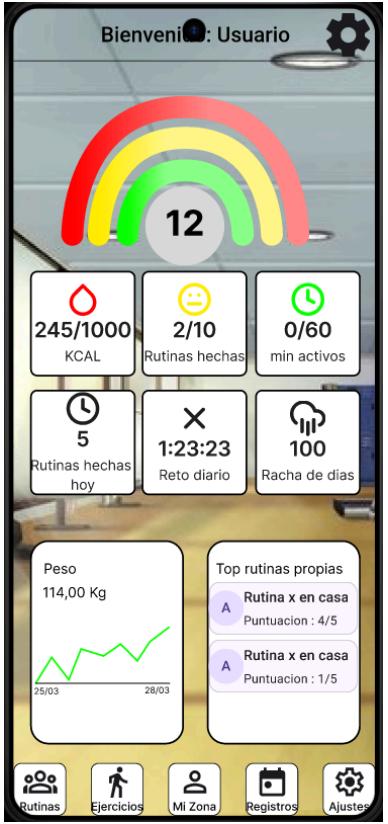
#### 4.5 pantalla crear rutina

	<p>Pantalla para crear rutinas nuevas y subirla para que otros usuarios puedan hacerlas y la puedan puntuar, el equipo se autocalcula según el ejercicio que meta (EJ: si para curls de bíceps necesitas una mancuerna el equipo saldrá mancuerna) el usuario podrá elegir un ícono y a futuro podrá poner una imagen suya pero antes tendrá que pasar por revisión del equipo técnico</p>
Imagen: Pantalla de crear rutina.	

#### 4.6 pantalla realizar ejercicio

	<p>Pantalla donde el usuario tendrá un contador para ver cuanto tiempo tarda en realizar todas los ejercicios que contiene la rutina y si tarda un tiempo razonable tendrá x2 de estadísticas, esta características se implementará a futuro</p>
Imagen: Pantalla de realizar ejercicio.	

#### 4.7 Pantalla usuario Mi zona



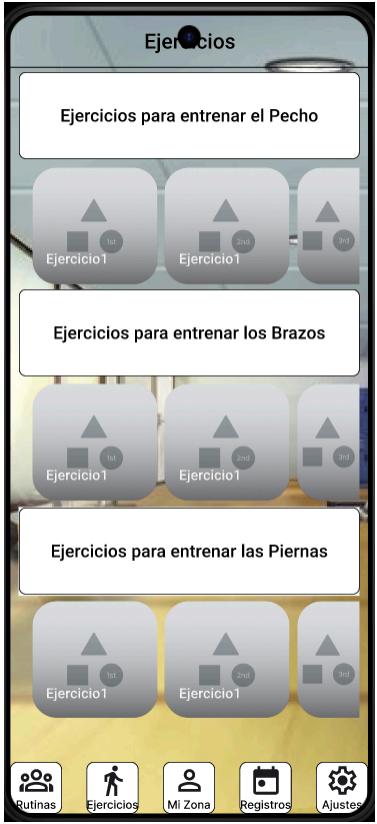
Pantalla donde podrá ver el usuario sus estadísticas, su peso, completar un reto diario de un ejercicio aleatorio, ver sus rutinas creadas.

Imagen: Pantalla de mi zona.

#### 4.8 reto diario

	<p>Pantalla que le indica al usuario el tiempo restante y el ejercicio a realizar junto con su repeticiones, después de darle a completado solo se altera la fecha donde hizo su último reto diario, completar el reto diario no suma a las estadísticas</p>
Imagen: Pantalla de reto diario.	

#### 4.9 elegir ejercicios



The image shows a mobile application interface titled "Ejercicios". The screen is divided into three main sections: "Ejercicios para entrenar el Pecho", "Ejercicios para entrenar los Brazos", and "Ejercicios para entrenar las Piernas". Each section contains three exercise cards, each labeled "Ejercicio1" and showing a sequence of three icons (triangle, square, circle) with sub-labels "1st", "2nd", and "3rd". At the bottom of the screen are five navigation icons: "Rutinas" (person icon), "Ejercicios" (person walking icon), "Mi Zona" (person icon), "Registros" (calendar icon), and "Ajustes" (gear icon).

Pantalla donde el usuario puede elegir ejercicios, catalogados por grupo muscular, para crear su nueva rutina.

Imagen: Pantalla de elegir ejercicios.

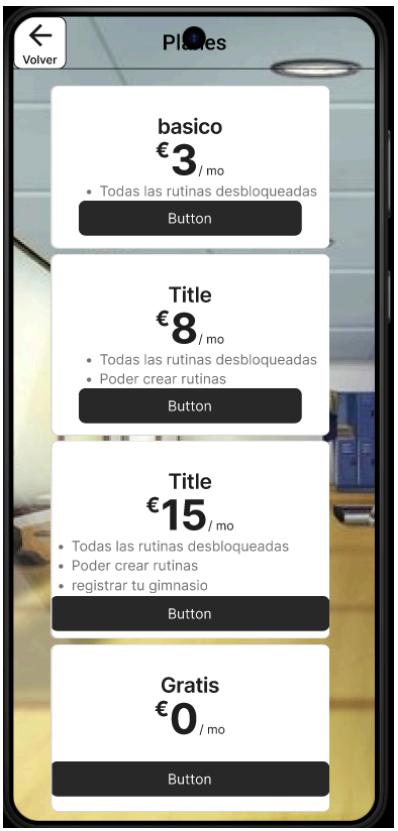
#### 4.10 Configuración



Pantalla donde el usuario puede poner o cambiar su plan de pago premium, el premium será implementado a futuro, registrar su propio gimnasio, característica a futuro y actualizar su cuenta característica a futuro

Imagen: Pantalla de configuración.

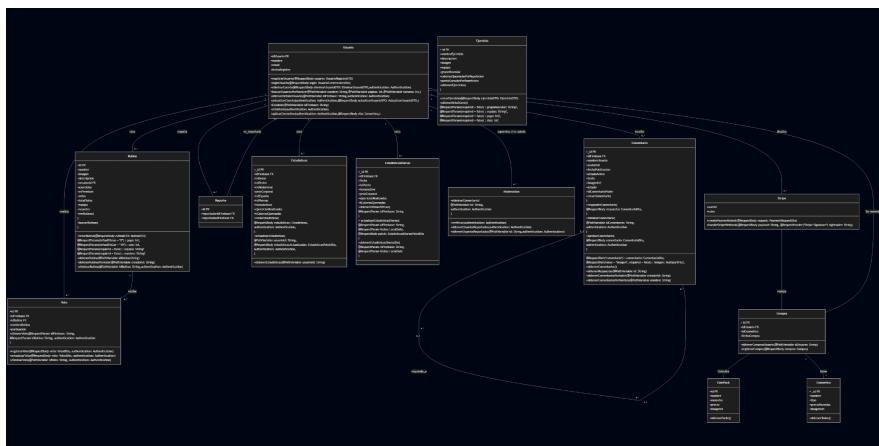
#### 4.11 Planes de pagos

	<p>Pantalla donde el usuario puede contratar o tener un plan de pago junto con sus ventajas dentro de la app, característica a futuro</p>
Imagen: Pantalla de planes de pagos.	

## 5. Diseño Funcional

## 5.1 Entidad Relación

## Imagen: Entidad Relación.



## Imagen completa

## 5.2 UsuariosController

Este controlador, orienta a toda la gestión, creación y manipulación de usuarios.

Este controlador usa la siguiente tabla:

### Usuario:

Tabla: Usuario.

Campo	Tipo	Descripción
idFirebase	String	Id principal que nos los otorga FireBase Auth.
sexo	String	Es el sexo del usuario y puede ser "M" Mujer, "H" Hombre u "O" Otro.
fechaNacimiento	LocalDate	Es la fecha de nacimiento del usuario.
nombre	String	Es el nombre del usuario.
correo	String	Es el correo del usuario.
FechaUltimoReto	LocalDate	Indica la última vez que el usuario hizo el reto diario.
gimnasiold	ObjectId?	Indica el Id del gimnasio donde está apuntado el usuario, si es null significa que no está apuntado a uno.
avatar	String	Indica la Uri de la foto avatar local en el dispositivo
esPremium	Boolean	Indica si es True es Premium y si es False no lo es.
perfilPublico	Boolean	Indica si es True el perfil es público y si es False el perfil es privado.
reportes	Int	Indica el número de reportes que ha recibido por parte de los usuarios.
rol	String	Indica el rol que tiene, si es "user" es un rol normal, en cambio si es "admin" es administrador.
monedas	Int	Indica el número de monedas que tiene el usuario
indumentaria	Indumentaria	Indica las prendas que tiene equipadas el usuario

Este controlador usa las siguientes entidades:

### UsuarioRegistroDTO:

Tabla: UsuarioRegistroDTO.

Campo	Tipo	Descripción
sexo	String	Puede ser "M"(Mujer), "H"(Hombre) o "O"(otro)
fechaNacimiento	LocalDate	La fecha de nacimiento del usuario
nombre	String	Nombre del usuario
correo	String	Correo del usuario
contrasena	String	Contraseña del usuario(NO se guarda en mongo)

**UsuarioregistradoDto:**

Tabla: UsuarioregistradoDto.

Campo	Tipo	Descripción
nombre	String	Nombre del usuario
correo	String	Correo del usuario

**UsuarioCredencialesDto:**

Tabla: UsuarioCredencialesDto.

Campo	Tipo	Descripción
correo	String	Correo del usuario que va a iniciar sesión
contrasena	String	Contraseña del usuario que va a iniciar sesión

**UsuarioLoginDto:**

Tabla: UsuarioLoginDto.

Campo	Tipo	Descripción
nombre	String	Nombre del usuario que ha iniciado sesión
token	String	JWT de firebase Auth que nos retorna al iniciar sesión

**EliminarUsuarioDTO:**

Tabla: EliminarUsuarioDTO.

Campo	Tipo	Descripción
correo	String	Correo del usuario que a ser eliminada su cuenta

**UsuarioBusquedaDto:**

Tabla: UsuarioBusquedaDto.

Campo	Tipo	Descripción
idFirebase	String	Id de firebase que está guardado en mongoDB
nombre	String	Nombre del usuario
sexo	String	Indica si el usuario es Hombre,Mujer u otro
esPremium	Boolean	True es premium, false es usuario normal
avatar	String	indica una uri creada, que se traduce a una foto de avatar local del móvil

**UsuarioInformacionDto:**

Tabla: UsuarioInformacionDto.

Campo	Tipo	Descripción
idFirebase	String	Id de firebase que está guardado en mongoDB
nombre	String	Nombre del usuario
correo	String	Correo del usuario
sexo	String	Indica si el usuario es Hombre,Mujer u otro
esPremium	Boolean	True es premium, false es usuario normal
avatarUrl	String	indica la uri del avatar
fechaUltimoReto	LocalDate	indica la fecha de la última vez que hizo el reto
estadisticas	EstadisticasDto	son las estadísticas del usuario
countRutinas	Long	indica las rutinas que ha creado el usuario
countComentarios	Long	indica cuántos comentarios ha creado el usuario
countVotos	Long	Indica cuántos votos ha realizado el usuario
monedas	Int	Indica cuántas monedas tiene el usuario
indumentaria	Indumentaria	Son los cosméticos que tiene equipado el usuario

**Indumentaria:**

Tabla: Indumentaria.

Campo	Tipo	Descripción
colorPiel	String	Uri que indica una foto de unos brazos
camiseta	String	Uri que indica una foto de una camiseta
tenis	String	Uri que indica una foto de unos Tenis
pantalon	String	Uri que indica una foto de unos pantalones

**EstadisticasDto:**

Tabla: EstadisticasDto.

Campo	Tipo	Descripción
idFirebase	String	Id de firebase que está asociada esa estadística
lvlBrazo	Double	Indica el nivel de brazo que tiene el usuario en total.
lvlAbdominal	Double	Indica el nivel de abdominales que tiene el usuario en total.
lvlEspalda	Double	Indica el nivel de espalda que tiene el usuario en total.
lvlPiernas	Double	Indica el nivel de piernas que tiene el usuario en total.
ejerciciosRealizados	Int	Indica los ejercicios realizados que tiene el usuario en total.
kCaloriasQuemadas	Double	Indica las KCAL quemadas que tiene el usuario en total.

**ActualizarUsuarioDTO:**

Tabla: ActualizarUsuarioDTO.

Campo	Tipo	Descripción
correo	String	Correo del usuario.
nombre	String	Nombre del usuario
sexo	String	Sexo del usuario
fechaNacimiento	LocalDate	Indica la fecha de nacimiento del usuario
perfilPublico	Boolean	True perfil público, false perfil privado
avatar	String	Indica si el avatar del usuario

Este controlador tiene los siguientes endpoint

### **5.2.1 /v1/usuarios**

Tabla: endpoints Usuarios.

Método	Ruta	Descripción	Request Body	Response
POST	/registrarse	Permite al usuario registrarse en Rutify.	UsuarioRegistroDTO	UsuarioRegistradoDto
POST	/acceder	Permite al usuario acceder a Rutify.	UsuarioCredencialesDto	UsuarioLoginDto
DELETE	/eliminar	Permite al usuario eliminar su cuenta.	EliminarUsuarioDTO	
GET	/buscar/{nombre}/{pagina}/{tamano}	Permite al usuario buscar a otros usuarios, mediante página de spring boot.	-	List<UsuarioBusquedaDto>
GET	/detalle/{idFirebase}	Permite al usuario ver los detalles de otro usuario, si el perfil está en público.	-	UsuarioInformacionDto
PUT	/actualizar	Permite al usuario actualizar información de su cuenta.	ActualizarUsuarioDTO	UsuarioActualizadoDto
GET	/esAdmin/{idFirebase}	Verifica si el usuario tiene rol de administrador.	-	Boolean
POST	/reto-diario	Marca al usuario como que ha completado el reto diario.	-	Boolean
PUT	/avatar/cosmetico	Aplica el cosmético al avatar seleccionado por el usuario.	Cosmetico	-

### 5.2.2 usuario se registra

#### Descripción funcional

Este endpoint POST /registrarse permite a un nuevo usuario registrarse en la plataforma Rutify. La lógica de negocio incluye validaciones de datos, creación de usuario en Firebase Authentication, almacenamiento en Firestore y persistencia en MongoDB.

#### Validaciones

**Nombre:** no puede estar vacío.

**Correo:** debe tener formato válido y no estar registrado.

**Contraseña:** mínimo 6 caracteres.

**Edad:** mínimo 16 años.

**Sexo:** debe ser "H", "M" u "O".

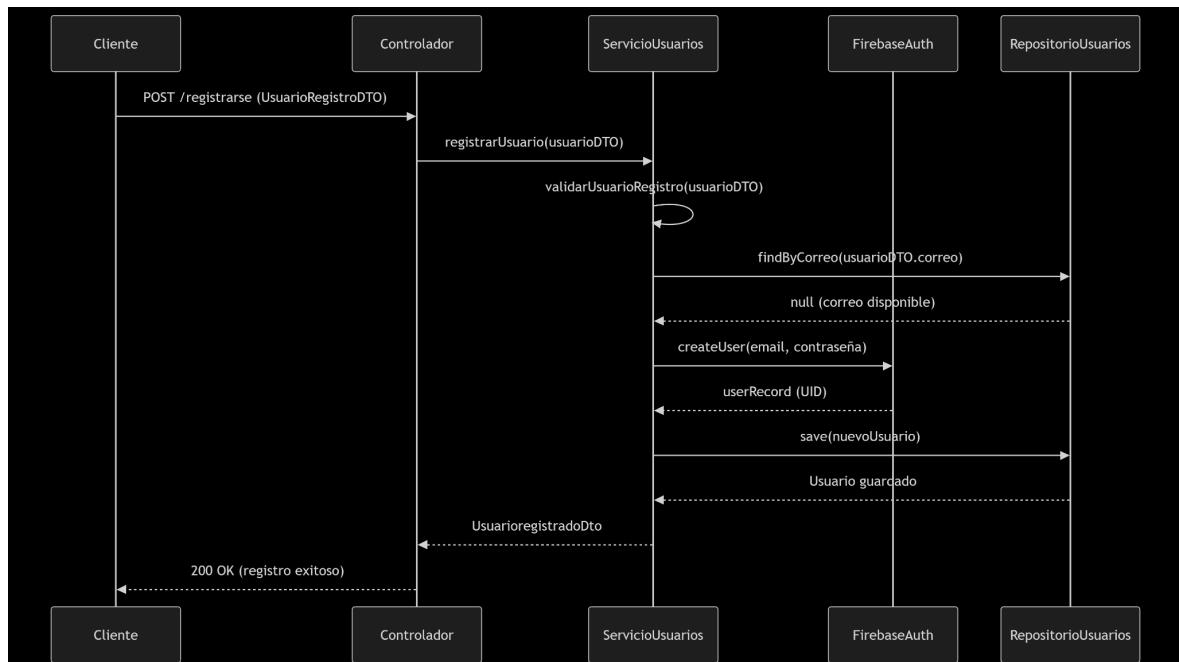
#### Ejemplos de Request JSON

Tabla: Json E/S usuario se registra

Json entrada	Json salida
<pre>{   "sexo": "H",   "edad": 25,   "nombre": "Lorem Ipsum",   "correo": "LoremIpsum@example.com",   "contrasena": "Lorem Ipsum" }</pre>	<pre>{   "nombre": "Lorem Ipsum",   "correo": "LoremIpsum@example.com" }</pre>

#### Diagrama de flujo

Diagrama flujo: usuario se registra.



Como vemos firebase auth se encarga de proveernos el id Firebase y de la seguridad por le JWT.

### 5.2.3 usuario inicia sesión

#### Descripción funcional

Este endpoint POST /acceder permite al usuario iniciar sesión en la plataforma Rutify.

La lógica de negocio incluye, validación de las credenciales en Firebase Authentication, obtención del nombre del usuario desde Firestore, generación de un token de sesión JWT proporcionado por Firebase, devolución de un DTO con el nombre del usuario y su token.

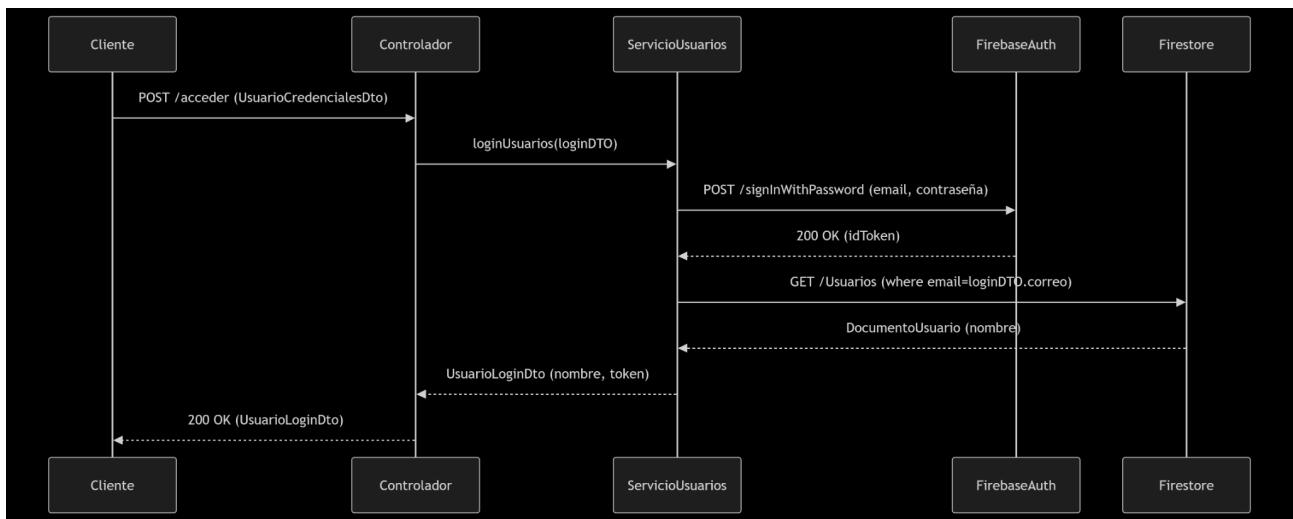
#### Ejemplos de Request JSON

Tabla: Json E/S usuario inicia sesión.

Json entrada	Json salida
{ "correo": "lorem.ipsum@example.com", "contrasena": "Lorem123" }	{ "nombre": "Lorem Ipsum", "token": "eXVCJ9..." }

#### Diagrama de flujo

Diagrama de flujo: usuario inicia sesión.



Como vemos nuestro inicio de sesión se valida con JWT de firebase, que se encarga de la seguridad y cifrado, en clientes móviles este endpoint no lo usamos, usamos el login de la librería de firebase directamente.

### 5.2.4 usuario elimina su cuenta

#### Descripción funcional

Este endpoint `DELETE /eliminar` permite a un usuario autenticado eliminar su cuenta de la plataforma Rutify, siempre que su UID coincida con el dueño de la cuenta que desea eliminar, verificación de identidad mediante el token JWT proporcionado en la autenticación, búsqueda del usuario por su correo en Firestore, verificación de permisos con respecto al UID del usuario autenticado, eliminación del usuario tanto de Firestore como de MongoDB, devolución de una confirmación con el correo eliminado

#### Validaciones

El usuario con ese correo debe existir.

El usuario autenticado debe tener permisos para eliminarlo.

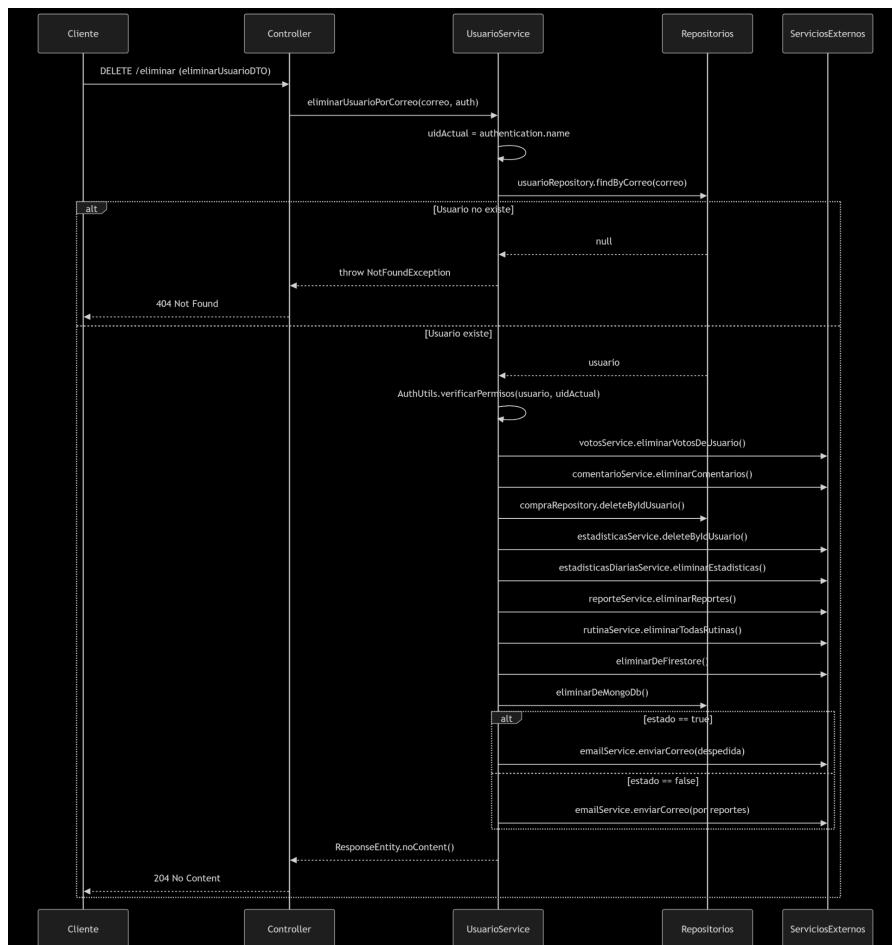
#### Ejemplos de Request JSON

Tabla: Json E/S usuario elimina su cuenta.

Json entrada	Json salida
{ "correo": "lorem.ipsum@example.com"}	

#### Diagrama de flujo

Diagrama de flujo: usuario elimina su cuenta.



Como vemos borramos toda información del usuario al instante, junto con firebase.

### 5.2.5 usuario busca a más usuarios

#### Descripción funcional

Este endpoint GET /buscar/{nombre}/{pagina}/{tamano} permite a un usuario buscar a otros usuarios dentro de Rutify según una coincidencia parcial con su nombre, búsqueda insensible a mayúsculas/minúsculas en MongoDB usando expresión regular (findByNombreContains), filtrado para incluir solo perfiles públicos, mapeo a un DTO que contiene información relevante para la vista de resultados.

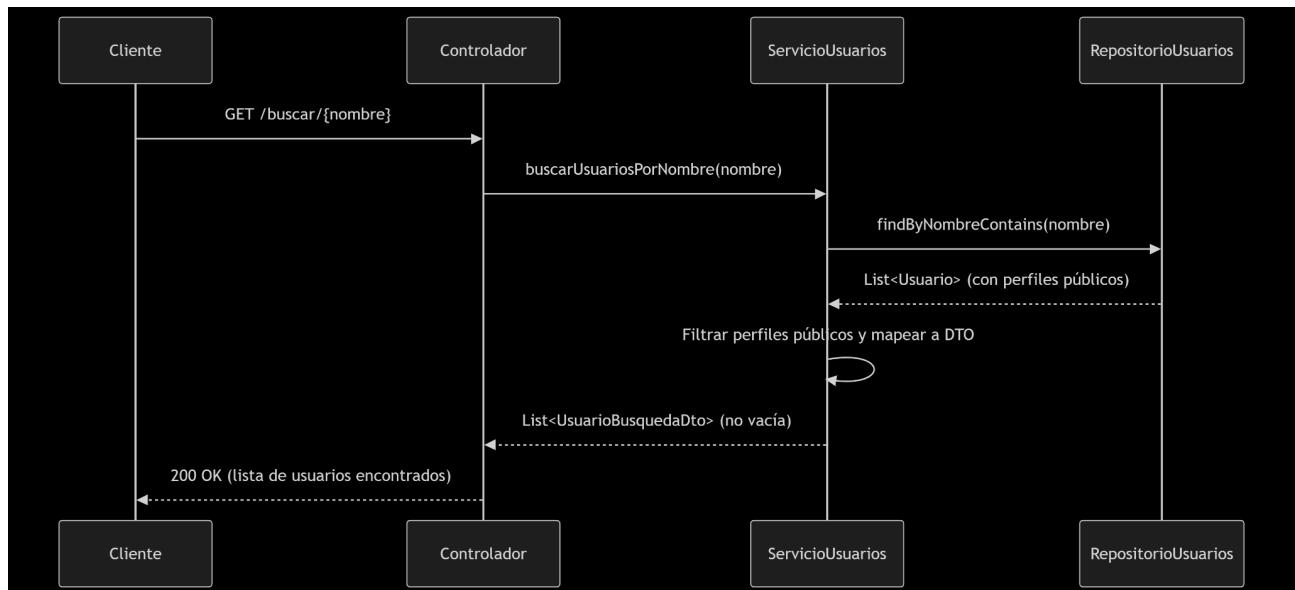
#### Ejemplos de Request JSON

Tabla: Json E/S usuario busca a más usuarios.

parámetro entrada	Json salida
/buscar/Lorem	[ { "idFirebase": "lorem123", "nombre": "Lorem Ipsum", "sexo": "H", "esPremium": true, "avatar": "https://example.com/avatar1.png" },... ]

#### Diagrama de flujo

Diagrama de flujo: usuario busca a más usuarios.



Como vemos retornamos la lista vacía o llena según lo que el usuario escriba.

### 5.2.6 usuario obtiene detalles de usuario

#### Descripción funcional

Este endpoint GET /detalle/{idFirebase} permite consultar los detalles completos de un usuario de Rutify, incluyendo estadísticas asociadas, se busca al usuario en MongoDB usando su id Firebase, se consultan las estadísticas correspondientes, si el perfil del usuario es público, se devuelve la información, si el perfil es privado, se devuelve un error 403 FORBIDDEN o 401 UNAUTHORIZED según la identidad del solicitante.

#### Validaciones

El usuario con ese idFirebase debe existir.

El perfil debe ser público o debe ser el propietario autenticado.

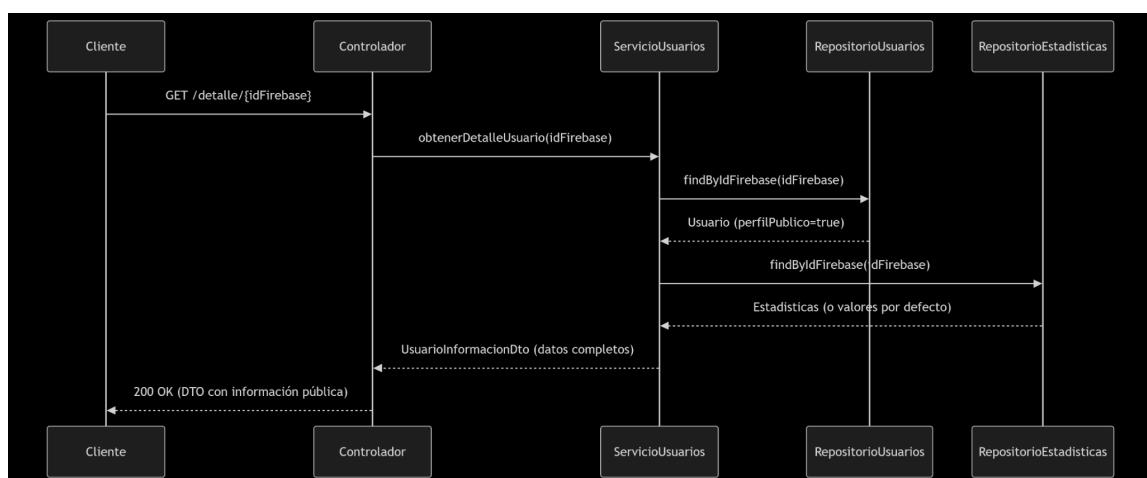
#### Ejemplos de Request JSON

Tabla: Json E/S usuario obtiene detalles de usuario.

Parámetro entrada	Json salida
/detalle/lorem123	{"idFirebase": "lorem123", "nombre": "Lorem Ipsum", "correo": "lorem ipsum@example.com", "sexo": "H", "esPremium": true, "avatarUrl": "https://example.com/avatar.png", "estadisticas": {"nivelPiernas": 3.5, "nivelBrazos": 2.0, "nivelPecho": 4.0, "nivelEspalda": 3.0, "diasEntrenados": 50, "puntosTotales": 89.5}}

#### Diagrama de flujo

Diagrama de flujo: usuario obtiene detalles de usuario.



Como vemos devuelve el usuario si tiene el perfilPublico en true, si no devuelve un 401.

### 5.2.7 usuario actualiza su cuenta

#### Descripción funcional

Este endpoint PUT /actualizar permite que un usuario actualice su propia cuenta o que un administrador actualice la cuenta de otro usuario en Rutify.

Se localiza al usuario solicitante mediante el id Firebase del token y al usuario objetivo por su correo electrónico. Si el solicitante no es administrador, solo puede modificar su propia cuenta.

La información actualizada se valida y se guarda en MongoDB.

#### Validaciones

El usuario solicitante debe existir.

El usuario a modificar debe existir (según el correo).

Si el solicitante no es administrador, solo puede modificar su propia cuenta.

Se valida el contenido del DTO (validarActualizarUsuarioDTO).

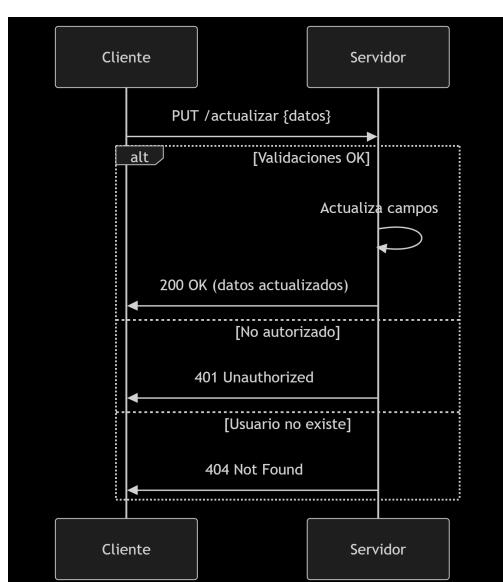
#### Ejemplos de Request JSON

Tabla: Json E/S usuario actualiza su cuenta.

Parámetro entrada	Json salida
{"correo": "lorem ipsum@example.com", "nombre": "Nuevo Nombre", "sexo": "M", "fechaNacimiento": "1998-05-10", "perfilPublico": true, "avatar": "https://example.com/avatar.png"}	{"correo": "lorem ipsum@example.com", "nombre": "Nuevo Nombre", "sexo": "M", "fechaNacimiento": "1998-05-10", "perfilPublico": true, "avatar": "https://example.com/avatar.png"}

#### Diagrama de flujo

Diagrama de flujo: usuario actualiza su cuenta.



Como vemos los datos se actualiza si tiene rol de administrador o es el mismo usuario.

### 5.2.8 usuario verifica si es admin

#### Descripción funcional

Este endpoint GET /esAdmin/{idFirebase} permite verificar si un usuario es administrador en la plataforma Rutify. Se busca al usuario por su id Firebase en MongoDB y se devuelve true si su rol es "admin", o false.

#### Validaciones

El usuario con el idFirebase especificado debe existir en la base de datos.

Si no existe, se lanza un error 404 (Usuario no encontrado)

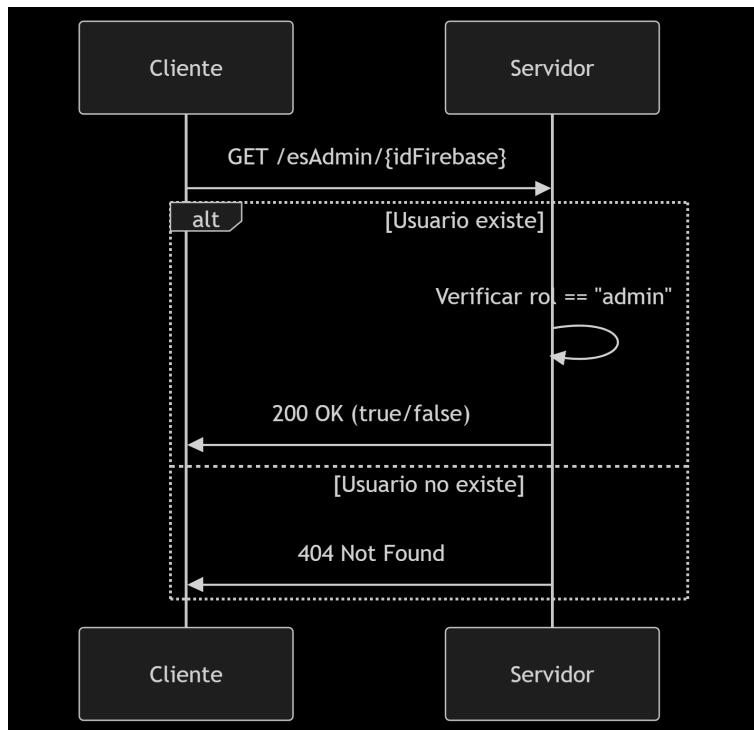
#### Ejemplos de Request JSON

Tabla: Json E/S usuario verifica si es administrador.

Parámetro entrada	Parámetro salida
idfirebase	true o false

#### Diagrama de flujo

Diagrama de flujo: usuario verifica si es administrador.



Como vemos esto solo comprueba el rol del usuario

### 5.2.9 usuario reto diario

#### Descripción funcional

Este endpoint POST /reto-diario permite que un usuario marque que ha completado el reto diario. Se busca al usuario autenticado por su idFirebase. Si el reto ya fue marcado hoy, devuelve true (indica que ya estaba marcado). Si no, actualiza la fecha del último reto al día actual, guarda el usuario y devuelve false (indica que se marcó por primera vez hoy).

#### Validaciones

El usuario autenticado debe existir.

Se compara la fecha del último reto con la fecha actual para determinar el estado.

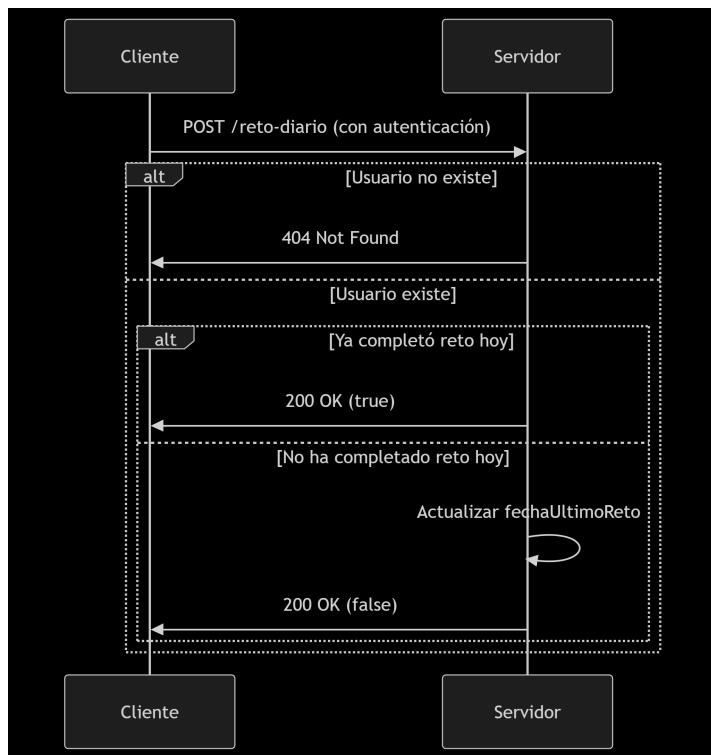
#### Ejemplos de Request JSON

Tabla: Json E/S usuario reto diario.

Parámetro entrada	Parámetro salida
authentication	true o false

#### Diagrama de flujo

Diagrama de flujo: usuario reto diario.



Como vemos el modo de ejecución es sencillo, se actualiza la fecha al día del reto y devuelve true o false.

### 5.2.10 usuario cambiar indumentaria

#### Descripción funcional

Este endpoint PUT /avatar/cosmetico permite que el usuario autenticado modifique la apariencia de su avatar aplicando un cosmético. El usuario debe enviar un objeto Cosmetico con el tipo (piel, camiseta, pantalón, tenis) y la URL de la imagen correspondiente. Se actualiza el campo correspondiente en la indumentaria del usuario y se guarda en la base de datos.

#### Validaciones

El usuario autenticado debe existir.

El tipo de cosmético (dto.tipo) debe ser uno de: piel, camiseta, pantalón, tenis.

Si el tipo es desconocido, se comete un error de validación.

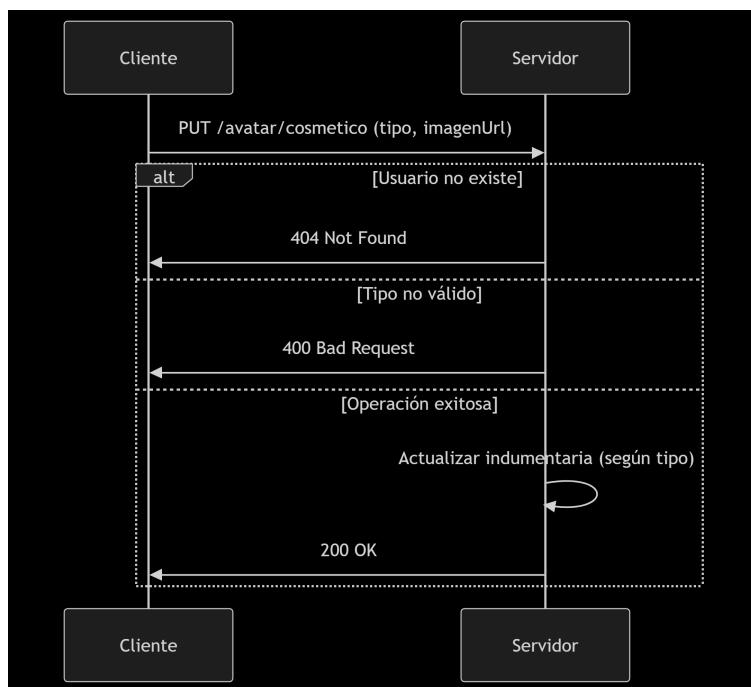
#### Ejemplos de Request JSON

Tabla: Json E/S usuario cambiar indumentaria.

Parámetro entrada	Parámetro salida
{"_Id":"2323232", "nombre": "Camiseta Roja", "tipo": "camiseta", "precioMonedas": 100, "imagenUrl": "https://exae.com/camiseta.png"}	-

#### Diagrama de flujo

Diagrama de flujo: usuario cambiar indumentaria.



Como vemos si el cosmético no es reconocido no se aplica

### 5.3 RutinaController

Este controlador, orienta a toda la gestión, visualización , creación y manipulación de rutinas.

Este controlador usa la siguiente tabla:

Rutina:

Tabla: Rutina.

Campo	Tipo	Descripción
id	String	Id principal.
nombre	String	Nombre de la rutina.
imagen	String	Url de la imagen de la rutina.
descripcion	String	Descripción de la rutina.
creadorId	String	Id de firebase del usuario que creó la rutina.
ejercicios	Map<String,Int>	Diccionario que contiene los id de los ejercicios como clave y las repeticiones a realizar de los mismos como valor.
esPremium	Boolean	Indica si la rutina es premium.
votos	Float	Es el valor del total de votos que ha obtenido.
totalVotos	Int	Es el valor de todos los votantes que han votado esa rutina
equipo	String	Indica el equipo que se necesita para realizar esta rutina.
reportes	Int	Indica el número de reportes que han otorgado los usuarios.

Este controlador usa las siguientes entidades:

RutinaBuscadorDto:

Tabla: RutinaBuscadorDto.

Campo	Tipo	Descripción
id	String	Id principal.
nombre	String	Nombre de la rutina.
imagen	String	Url de la imagen de la rutina.
descripcion	String	Descripción de la rutina.
cuantosEjercicios	Int	Indica el número de ejercicios que contiene la rutina.
votos	Float	Es el valor del total de votos que ha obtenido.
totalVotos	Int	Es el valor de todos los votantes que han votado esa rutina.
esPremium	Boolean	Indica si la rutina es premium.
equipo	String	Indica el equipo que se necesita para realizar esta rutina.

RutinaDTO:

Tabla: RutinaDTO.

Campo	Tipo	Descripción
id	String	Id principal.
nombre	String	Nombre de la rutina.
imagen	String	Url de la imagen de la rutina.
descripcion	String	Descripción de la rutina.
creadorId	String	Id de firebase del usuario que creó la rutina.
ejercicios	List<EjercicioDTO>	Es la lista de ejercicios que contiene la rutina,junto con sus repeticiones.
equipo	String	Indica el equipo que se necesita para realizar esta rutina.
votos	Float	Es el valor del total de votos que ha obtenido.
totalVotos	Int	Es el valor de todos los votantes que han votado esa rutina.
esPremium	Boolean	Indica si la rutina es premium.

EjercicioDTO:

Tabla: EjercicioDTO.

Campo	Tipo	Descripción
id	String	Id principal.
nombreEjercicio	String	Nombre del ejercicio.
descripcion	String	Descripción del ejercicio.
imagen	String	Url de la imagen del ejercicio.
equipo	String	Indica el equipo que se necesita para realizar este ejercicio.
grupoMuscular	String	Indica a qué grupo muscular va enfocado este ejercicio.
caloriasQuemadasPorRepeticion	Double	Es las calorías que quemas cada vez que haces una repetición.
puntoGanadosPorRepeticion	Double	Es la cantidad de experiencias que consigues al hacer una repetición.
cantidad	Int	Es la cantidad de repeticiones a realizar

### 5.3.1 /v1/rutinas

Este controlador tiene los siguientes endpoint

Tabla: endpoints Rutina.

Método	Ruta	Descripción	Request Body	Response
POST	/crear	Permite al usuario crear una rutina en Rutify.	RutinaDTO	RutinaDTO
GET	/verRutinas	Permite al usuario visualizar las rutinas creadas mediante paginación	-	List<RutinaBuscadorDto>
GET	/buscarRutinas	Permite al usuario buscar rutinas mediante su nombre parcial	-	List<RutinaBuscadorDto>
GET	/{{idRutina}}	Obtiene una rutina mediante su Id	-	RutinaDTO
GET	/autor/{{creadorId}}	Obtiene todas las rutinas mediante el id Firebase del autor	-	List<RutinaBuscadorDto>
DELETE	/eliminar/{{idRutina}}	elimina una rutina si es el mismo auto o un administrador	-	-

### 5.3.2 usuario crea una rutina

#### Descripción funcional

Este endpoint permite crear una nueva rutina personalizada en Rutify. El cliente envía un objeto RutinaDTO con la información de la rutina y su lista de ejercicios. Se validan los datos, se procesan los ejercicios para asociar sus IDs correctos y luego se guarda la rutina en la base de datos. Se devuelve la rutina enviada.

#### Validaciones

Los datos de la rutina deben ser válidos.

Cada ejercicio debe tener un id o un nombreEjercicio válido.

La lista de ejercicios no puede estar vacía.

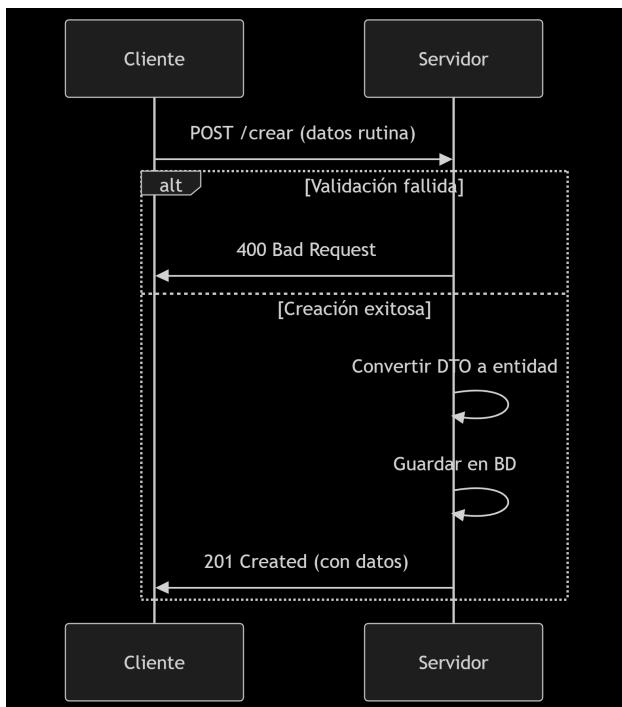
#### Ejemplos de Request JSON

Tabla: Json E/S usuario crea una rutina.

Json entrada	Json salida
{"nombre": "Rutina Fuerza Total", "imagen": "https://example.com/rutina.png", "descripcion": "Entrenamiento para fuerza", "creadorId": "uid123", "ejercicios": [{"id": "ej123", "nombreEjercicio": "Press banca", "cantidad": 4}], "equipo": "mancuernas", "esPremium": true}	{"nombre": "Rutina Fuerza Total", "imagen": "https://example.com/rutina.png", "descripcion": "Entrenamiento para fuerza", "creadorId": "uid123", "ejercicios": [{"id": "ej123", "nombreEjercicio": "Press banca", "cantidad": 4}], "equipo": "mancuernas", "esPremium": true}

#### Diagrama de flujo

Diagrama flujo: usuario crea una rutina.



Como vemos el usuario puede crear una rutina fácilmente.

### 5.3.3 usuario crea ver rutinas

#### Descripción funcional

Este endpoint GET /verRutinas permite obtener una lista paginada de rutinas de entrenamiento disponibles en Rutify. Las rutinas se pueden filtrar opcionalmente por el tipo de equipo requerido (equipo) mediante una búsqueda no sensible a mayúsculas/minúsculas.

Se utiliza una lógica de paginación que permite controlar cuántas rutinas se obtienen por página y desde qué página empezar, facilitando la carga progresiva de datos en el frontend.

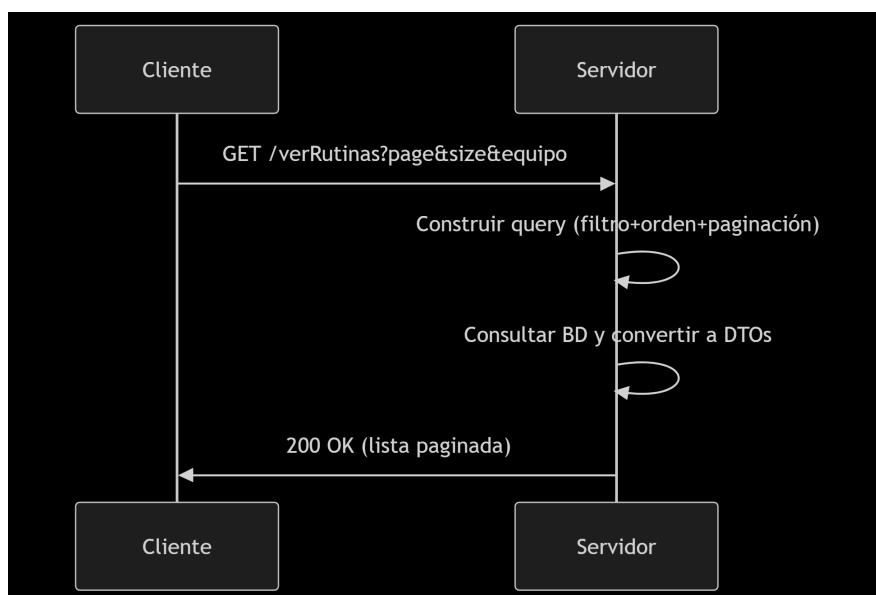
#### Ejemplos de Request JSON

Tabla: Json E/S usuario ver rutinas.

Json entrada	Json salida
page: Número de página que se desea obtener. size: Número de rutinas que se desea recuperar por página. equipo: Texto para filtrar las rutinas por tipo de equipo.	[{"id": "rutina123", "nombre": "Fuerza con mancuernas", "imagen": "https://example.com/rutina.png", "descripcion": "Rutina enfocada en brazos y pecho", "equipo": "mancuernas", "esPremium": false}]

#### Diagrama de flujo

Diagrama flujo: usuario ver rutinas.



Como vemos si no envias nada por defecto te carga los 10 primeros ejercicios.

#### 5.3.4 usuario crea ver rutinas

##### Descripción funcional

El endpoint GET /buscarRutinas permite a los usuarios buscar rutinas de entrenamiento por nombre. La búsqueda no es sensible a mayúsculas/minúsculas y filtra las rutinas cuyo nombre comienza con el texto proporcionado.

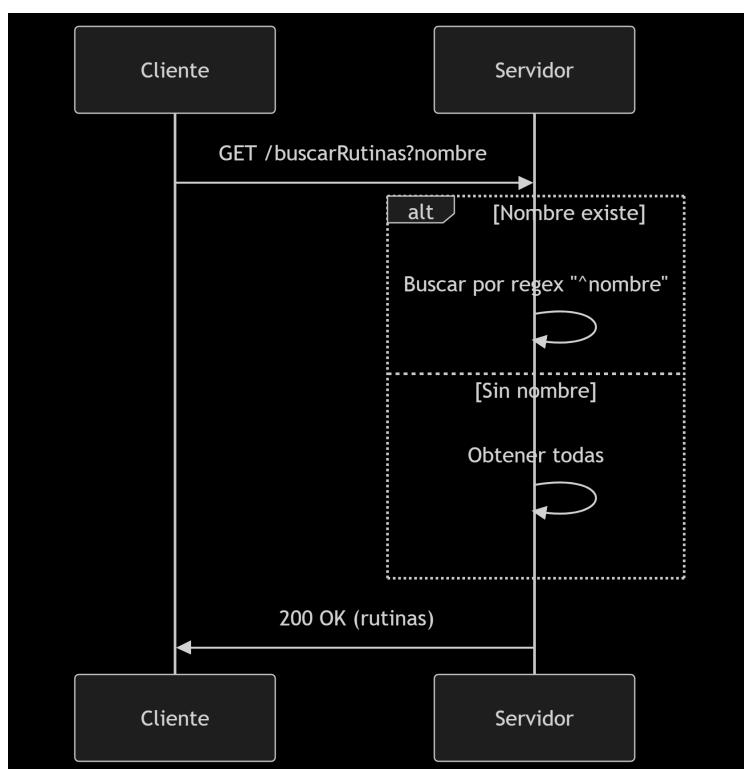
##### Ejemplos de Request JSON

Tabla: Json E/S usuario ver rutinas.

Json entrada	Json salida
nombre: Texto para filtrar las rutinas por nombre.	[{"id": "rutina123", "nombre": "Fuerza con mancuernas", "imagen": "https://ee.com/rutina.png", "descripcion": "Rutina en brazos y pecho", "equipo": "mancuernas", "esPremium": false}]

##### Diagrama de flujo

Diagrama flujo: usuario ver rutinas.



Como vemos nos retorna todas las rutinas cuyo nombre sea parcialmente igual.

### 5.3.5 usuario obtener detalles rutina

#### Descripción funcional

Este endpoint GET /{idRutina} permite obtener todos los detalles de una rutina específica mediante su identificador único.

#### Validaciones

**ID válido:** Se espera un idRutina no nulo ni vacío.

**Rutina existente:** Si no se encuentra la rutina, se lanza NotFoundException.

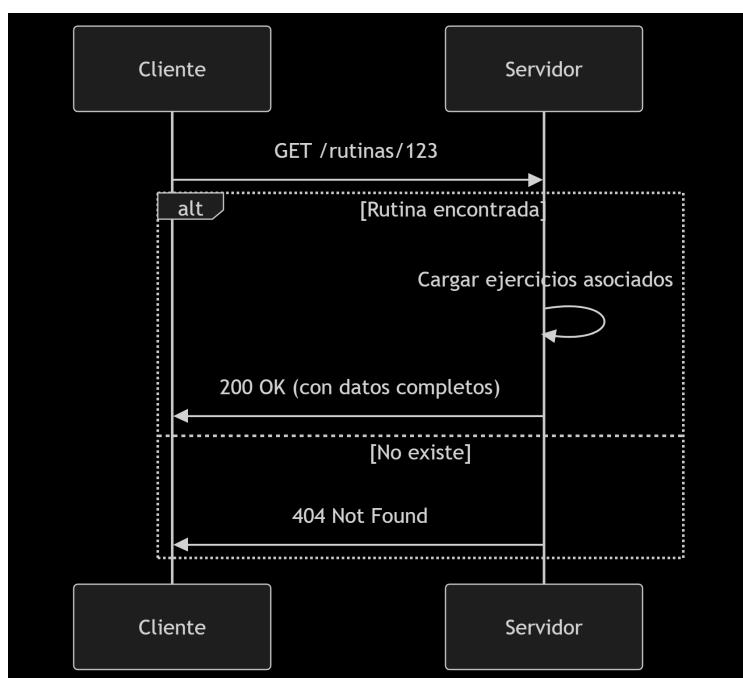
#### Ejemplos de Request JSON

Tabla: Json E/S obtener detalles rutina.

Json entrada	Json salida
	{"id": "rutina123", "nombre": "Fuerza con mancuernas", "imagen": "https://example.com/rutina.png", "descripcion": "Rutina en brazos y pecho", "creadorId": "usuario456", "equipo": "mancuernas", "votos": 4.5, "totalVotos": 20, "esPremium": false, "ejercicios": [{"id": "ej1", "nombreEjercicio": "Curl de bíceps", "cantidad": 3}]}

#### Diagrama de flujo

Diagrama flujo: obtener detalles de la rutina.



Como vemos el spring boot se encarga de traducir de map a list de ejercicios.

### 5.3.6 usuario obtener rutinas por autor

#### Descripción funcional

Este endpoint GET /autor/{creadorId} permite obtener todas las rutinas creadas por un usuario específico. Se utiliza para mostrar en el perfil del autor todas las rutinas que ha publicado.

#### Validaciones

creadorId: Si el ID está vacío o en blanco, lanza ValidationException.

Autor con rutinas: Si el autor no tiene rutinas, devuelve lista vacía.

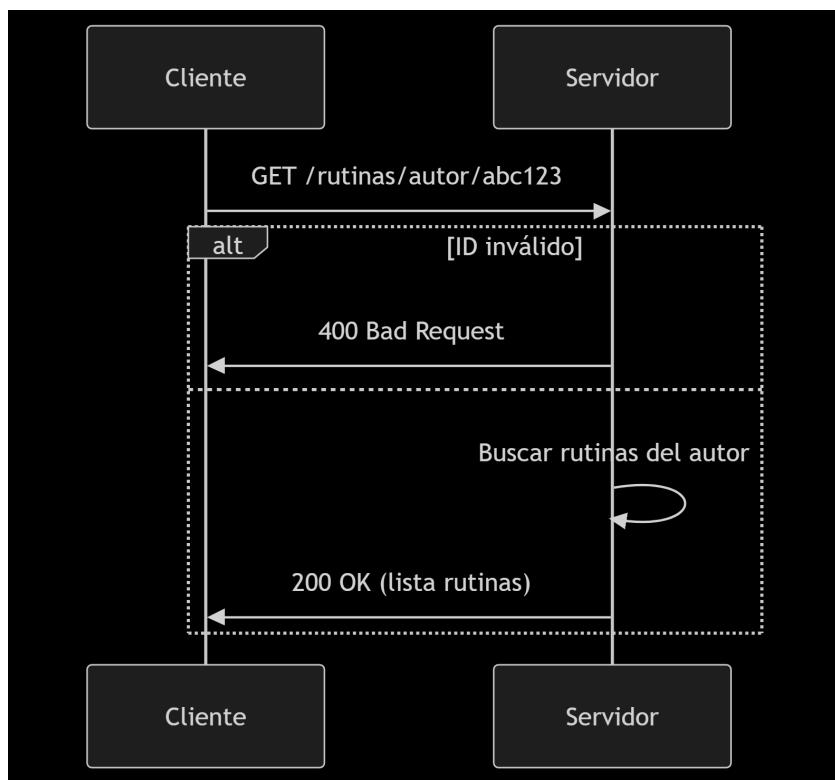
#### Ejemplos de Request JSON

Tabla: Json E/S usuario obtiene rutinas por autor.

Json entrada	Json salida
creadorId: ID del usuario autor de las rutinas.	[{"id": "rutina456", "nombre": "Cardio extremo", "imagen": "https://example.com/cardio.png", "descripcion": "Rutina intensa de cardio en casa", "equipo": "sin equipo", "esPremium": true}]

#### Diagrama de flujo

Diagrama flujo: usuario obtener rutinas por autor.



Como vemos o nos trae todo las rutinas o vacío o un 400 si el id no es válido o un 404 si no existe.

### 5.3.7 usuario elimina una rutina

#### Descripción funcional

Este endpoint `DELETE /eliminar/{idRutina}` permite eliminar una rutina específica creada por el usuario autenticado. También permite a un administrador eliminar cualquier rutina. En caso de eliminación por parte del administrador, se notifica al creador original de la rutina por correo electrónico.

#### Validaciones

**idRutina:** Si el ID está vacío o la rutina no existe, lanza `NotFoundException`.

**authentication.name:** Si el usuario no existe en la base de datos, lanza `NotFoundException`.

**Permisos:** Si el usuario no es el creador ni tiene rol de administrador, lanza `UnauthorizedException`.

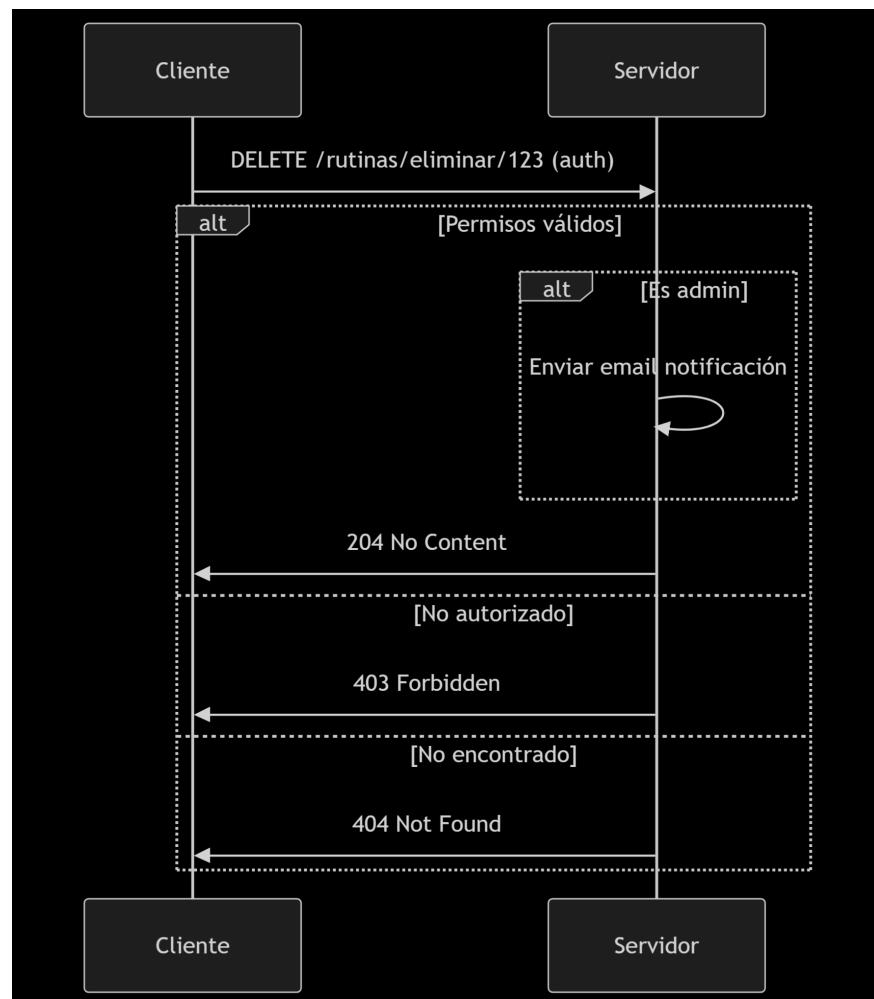
#### Ejemplos de Request JSON

Tabla: Json E/S usuario elimina una rutina.

Json entrada	Json salida
rutina789	204 No Content

#### Diagrama de flujo

Diagrama flujo: usuario elimina una rutina.



Como vemos si lo eliminar el mismo autor o un admin sale todo bien y se notifica en caso de admin.

## 5.4 VotosController

Este controlador, orienta a toda la gestión, creación y manipulación de votos.

Este controlador usa la siguiente tabla:

Voto:

Tabla: Votos.

Campo	Tipo	Descripción
id	String	Id principal.
idFirebase	String	Id que asocia el voto con el usuario.
idRutina	String	El id de la rutina que está siendo votada.
nombreRutina	String	El nombre de la rutina que está siendo votada
puntuacion	Float	La puntuación que recibe la rutina.

Este controlador usa las siguientes entidades:

VotodDto:

Tabla: VotodDto.

Campo	Tipo	Descripción
id	String	Id principal.
idFirebase	String	Id que asocia el voto con el usuario.
idRutina	String	El id de la rutina que está siendo votada.
nombreRutina	String	El nombre de la rutina que está siendo votada
puntuacion	Float	La puntuación que recibe la rutina.

#### **5.4.1 /v1/rutinas**

Este controlador tiene los siguientes endpoint

Tabla: endpoints Votos.

Método	Ruta	Descripción	Request Body	Response
POST	/registrar	Permite al usuario registrar un voto nuevo.	VotodDto	VotodDto
GET	/obtenerVoto	Obtiene el voto de un usuario, mediante su id firebase y el de la rutina	-	VotodDto
PATCH	/actualizar	Permite al usuario actualizar su voto ya existente.	VotodDto	VotodDto
DELETE	/eliminarVoto/{idVoto}	Permite al usuario eliminar su propio voto.	-	-
GET	/autor/{creadorId}	Obtiene todos los votos de un usuario	-	List<VotodDto>

#### 5.4.2 usuario registra voto

##### Descripción funcional

Este endpoint POST /registrar permite que un usuario autenticado registre una votación sobre una rutina específica. El voto incluye una puntuación del 1 al 5, y cada usuario solo puede votar una vez por rutina. Además, se actualiza el total acumulado de votos y la cantidad de votantes de la rutina.

##### Validaciones

**idFirebase:** Debe coincidir con el usuario autenticado. Si no, lanza UnauthorizedException.

**idRutina:** Debe existir en la base de datos.

**Voto duplicado:** Si el usuario ya votó esa rutina, lanza ConflictException.

**puntuación:** Se valida con validarVotos(). Debe estar en el rango permitido (por ejemplo, 1 a 5).

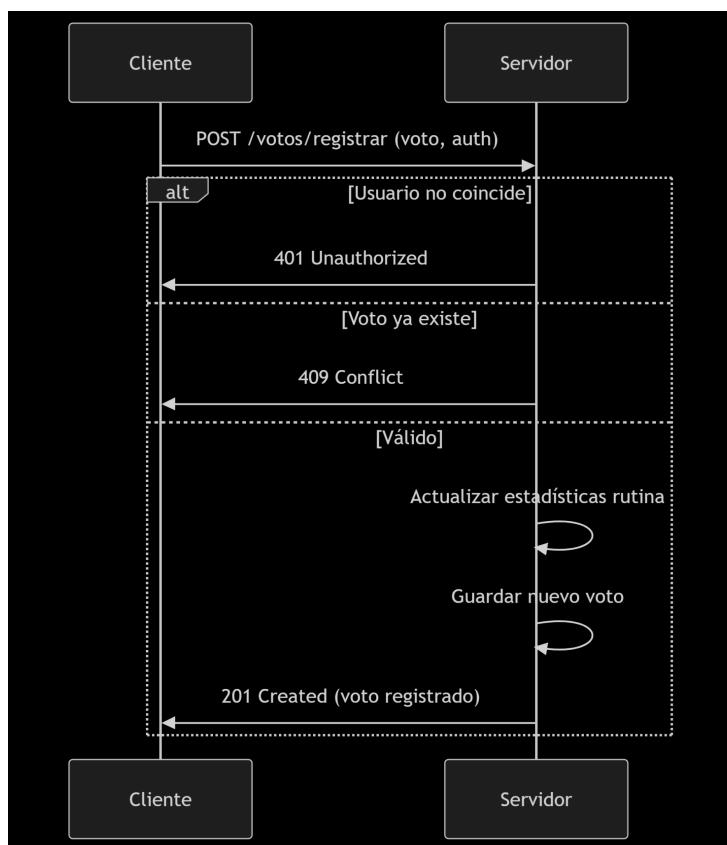
##### Ejemplos de Request JSON

Tabla: Json E/S registra voto

Json entrada	Json salida
{"idFirebase": "usuario123", "idRutina": "rutina456", "puntuacion": 4.5, "nombreRutina": "Cardio extremo"}	{"idFirebase": "usuario123", "idRutina": "rutina456", "puntuacion": 4.5, "nombreRutina": "Cardio extremo"}

##### Diagrama de flujo

Diagrama flujo: registra voto.



Como vemos, el usuario no podrá crear un voto a otra persona, aun siendo administrador.

#### 5.4.3 usuario obtiene votos

##### Descripción funcional

Este endpoint GET /obtenerVoto permite a un usuario autenticado consultar su voto registrado sobre una rutina específica. Sirve para mostrar al usuario si ya ha valorado una rutina y con qué puntuación.

##### Validaciones

**idFirebase:** Debe coincidir con el usuario autenticado. Si no, lanza UnauthorizedException.

**idRutina:** Debe ser válido.

**Voto:** Si no existe el voto para esa rutina por parte del usuario, lanza ValidationException.

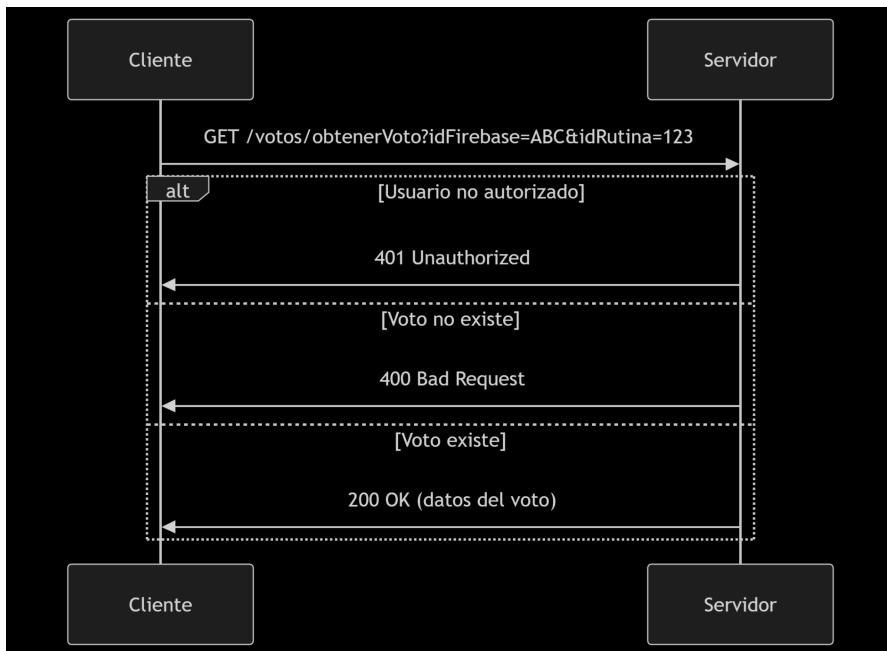
##### Ejemplos de Request JSON

Tabla: Json E/S usuario obtiene votos

Json entrada	Json salida
/obtenerVoto?idFirebase=usuario123&idRutina=rutina456	{"idFirebase": "usuario123", "idRutina": "rutina456", "puntuacion": 4.0, "nombreRutina": "Fuerza en casa"}

##### Diagrama de flujo

Diagrama flujo: usuario obtiene votos.



Como política de privacidad, tengo una lógica de negocio en la que los votos son privados.

#### 5.4.4 usuario actualiza su voto

##### Descripción funcional

Este endpoint PATCH /actualizar permite a un usuario autenticado modificar su voto previo registrado sobre una rutina. Se actualiza la puntuación y se ajustan los totales de votos y puntuaciones acumuladas en la rutina.

##### Validaciones

Se valida que el idFirebase del voto coincida con el usuario autenticado, si no, se lanza una UnauthorizedException.

Se valida que la puntuación esté dentro de los parámetros permitidos con la función validarVotos.

Se verifica que el voto exista; de no ser así, se lanza una excepción.

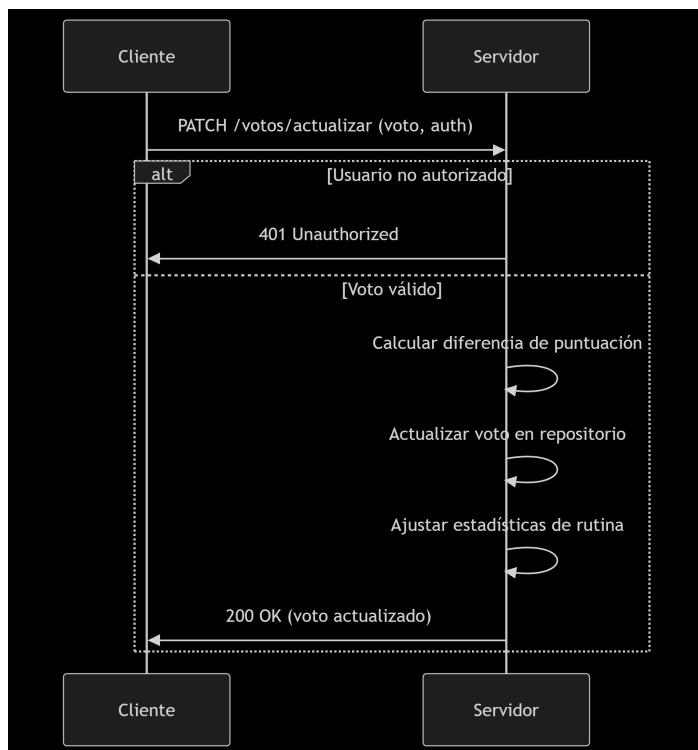
##### Ejemplos de Request JSON

Tabla: Json E/S actualiza su voto

Json entrada	Json salida
{"id": "voto789", "idFirebase": "usuario123", "idRutina": "rutina456", "puntuacion": 5.0, "nombreRutina": "Fuerza en casa"}	{"id": "voto789", "idFirebase": "usuario123", "idRutina": "rutina456", "puntuacion": 5.0, "nombreRutina": "Fuerza en casa"}

##### Diagrama de flujo

Diagrama flujo: actualiza su voto.



Como vemos, a la hora de actualizar también ajustar la puntuación en la rutina.

#### 5.4.5 usuario se registra

##### Descripción funcional

Este endpoint `DELETE /eliminarVoto/{idVoto}` permite a un usuario autenticado eliminar un voto previamente registrado sobre una rutina. Se actualizan los totales de votos y puntuaciones de la rutina correspondiente al eliminarse el voto.

##### Validaciones

**idVoto:** Se verifica que el voto exista en la base de datos. Si no se encuentra, se lanza una `ValidationException`.

**Autenticación:** Se valida que el voto pertenezca al usuario autenticado. Si no coincide, se lanza una `UnauthorizedException`.

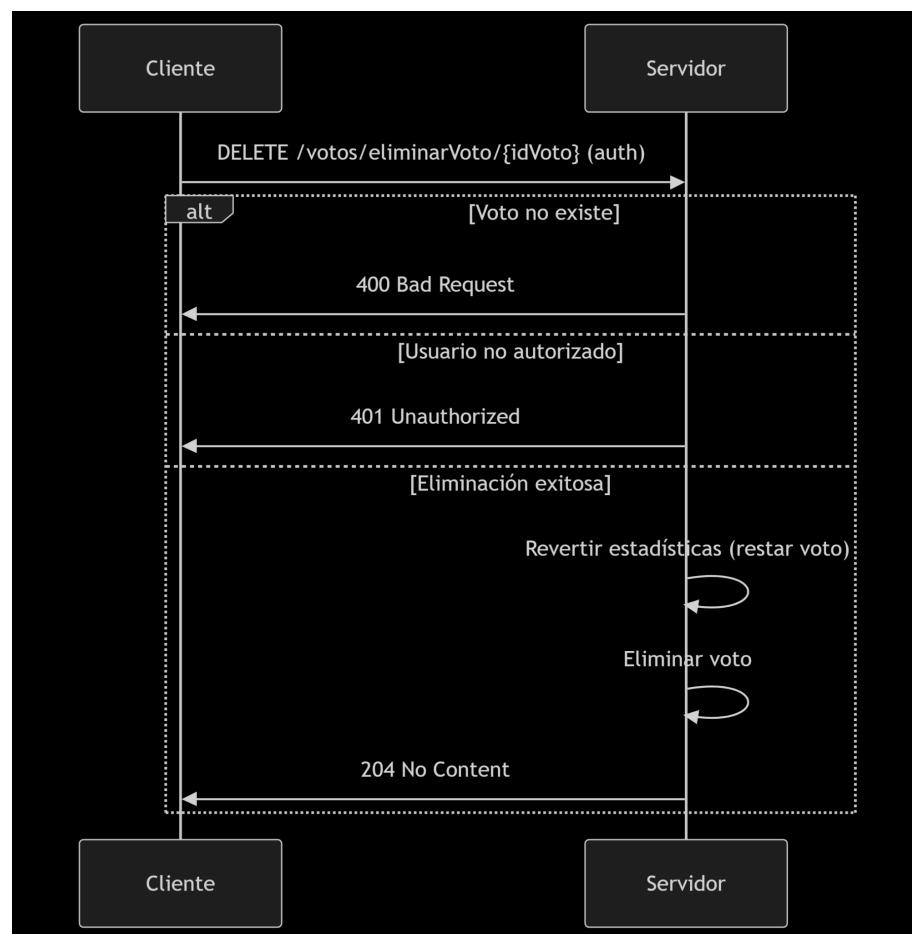
##### Ejemplos de Request JSON

Tabla: Json E/S usuario se registra

Json entrada	Json salida
-	204 No Content

##### Diagrama de flujo

Diagrama flujo: usuario se registra.



Como vemos a la hora de eliminar ajustamos la puntuación de la rutina.

#### 5.4.6 usuario obtiene votos por autor

##### Descripción funcional

Este endpoint GET /autor/{creadorId} permite obtener todos los votos realizados por un usuario específico. Se utiliza para que un usuario vea todas las valoraciones que ha realizado sobre distintas rutinas.

##### Validaciones

**creadorId:** No puede estar vacío ni en blanco. Si lo está, se lanza una ValidationException.

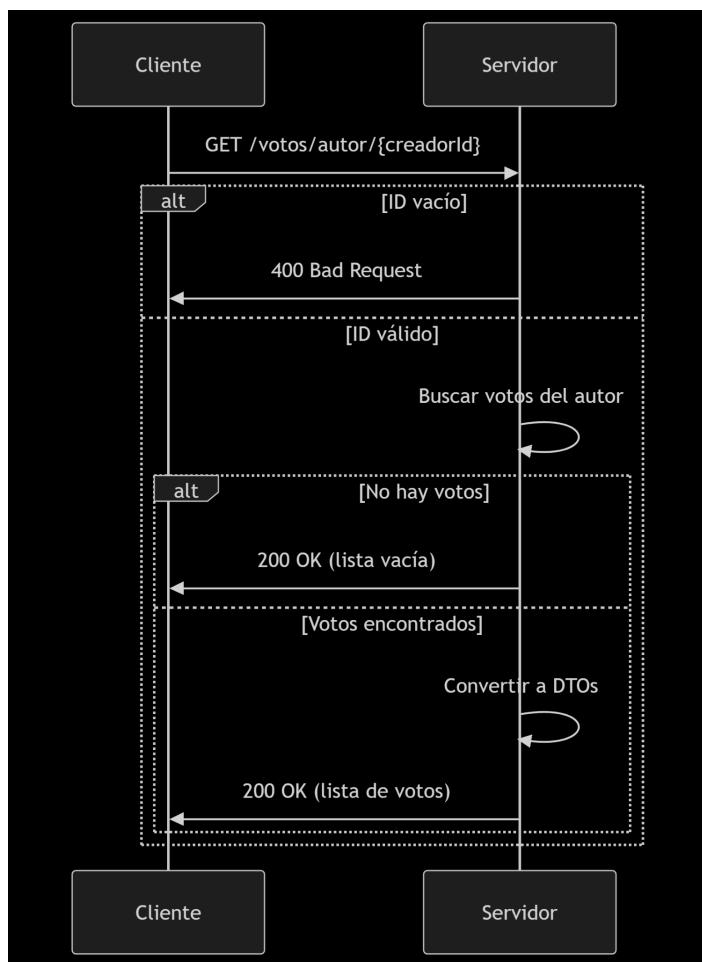
##### Ejemplos de Request JSON

Tabla: Json E/S usuario obtiene votos por autor

Json entrada	Json salida
usuario123	[{"idFirebase": "usuario123", "idRutina": "rutina456", "puntuacion": 5.0, "nombreRutina": "Piernas de acero"}]

##### Diagrama de flujo

Diagrama flujo: usuario obtiene votos por autor.



Como vemos si no tiene votos, manda una lista vacía.

## 5.5 StripeController

Este controlador, se encarga de gestionar las compras con tarjeta de crédito

Este controlador usa las siguientes entidades:

PaymentRequestDto:

Tabla: PaymentRequestDto.

Campo	Tipo	Descripción
userId	String	El id de firebase del usuario que le enviamos con los metadatos.
coins	Int	La cantidad de monedas que va a recibir el usuario

PaymentResponse:

Tabla: PaymentResponse.

Campo	Tipo	Descripción
clientSecret	String	Permite autenticar el pago con medios seguros como tarjetas o wallets.

### 5.5.1 /v1/pagos

Este controlador tiene los siguientes endpoint

Tabla: endpoints Rutina.

Método	Ruta	Descripción	Request Body	Response
POST	/v1/pagos	Permite a un usuario crear una intención de pago para adquirir monedas virtuales dentro de la app.	PaymentRequestDto	UsuarioregistradoDto
POST	/stripe-webhook	recibe notificaciones (webhooks) enviadas por Stripe cuando hay eventos relacionados con pagos (como la confirmación de un pago)	payload	String

### 5.5.1 usuario crea intención de pago con Stripe

#### Descripción funcional

Este endpoint POST /v1/pagos permite a un usuario crear una intención de pago para adquirir monedas virtuales dentro de la app. Utiliza la API de Stripe para generar un PaymentIntent y devolver su clientSecret, que se usará en el frontend para completar el pago.

#### Validaciones

**coins:** Debe ser suficiente para alcanzar al menos 0.50€ (50 monedas). Si no, lanza una IllegalArgumentException.

**userId:** Se incluye como metadato en la intención de pago para su posterior identificación.

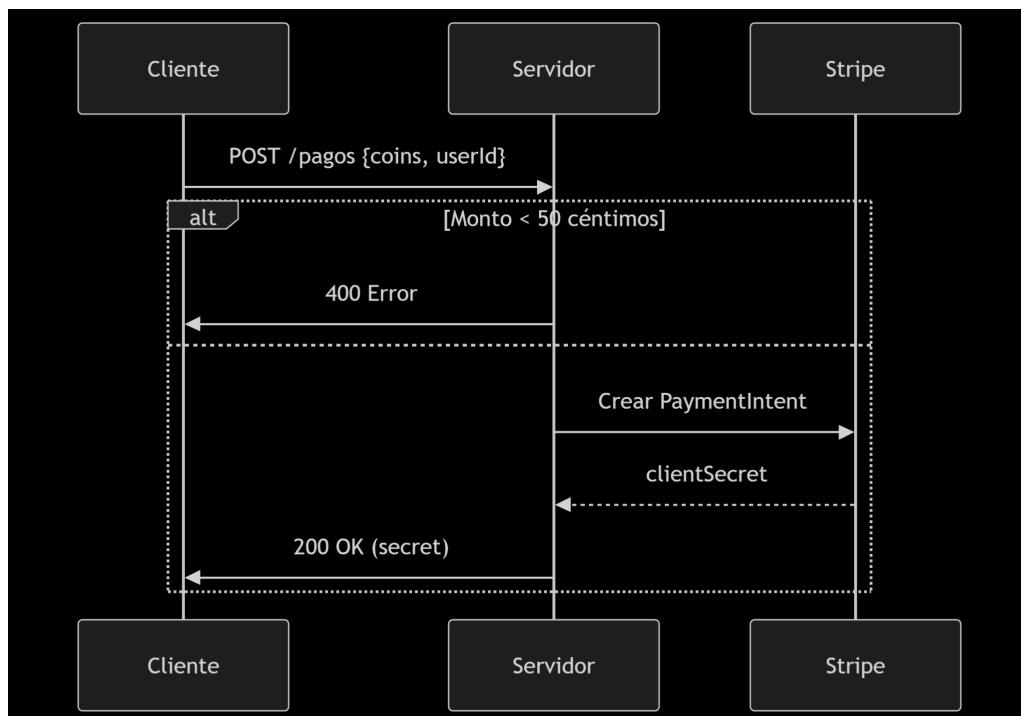
#### Ejemplos de Request JSON

Tabla: Json E/S usuario crea intención de pago con Stripe.

Json entrada	Json salida
{ "userId": "usuario123", "coins": 100 }	{ "clientSecret": "pi_3Nx12345secret_xyz" }

#### Diagrama de flujo

Diagrama flujo: usuario crea intención de pago con Stripe.



Como vemos delegó la responsabilidad en stripe, para el tema del pago.

### 5.5.2 Stripe webhook — manejo de notificaciones de pago

#### Descripción funcional

Este endpoint POST /stripe-webhook recibe notificaciones (webhooks) enviadas por Stripe cuando hay eventos relacionados con pagos (como la confirmación de un pago). Su función es validar la firma del webhook para garantizar la seguridad y luego procesar el evento, extrayendo la información de usuario y monedas para actualizar el balance en el sistema.

#### Validaciones

**Firma del webhook:** Se valida con la clave secreta (endpointSecret). Si la firma no es válida, se devuelve un error 400 (Firma inválida).

**Payload:** Se verifica que el payload contenga los campos metadata, con las claves userId y coins. Si falta alguna, se devuelve error 400 (Metadata faltante o incompleta).

**Errores inesperados:** Cualquier excepción durante el procesamiento devuelve error 500 con mensaje de error.

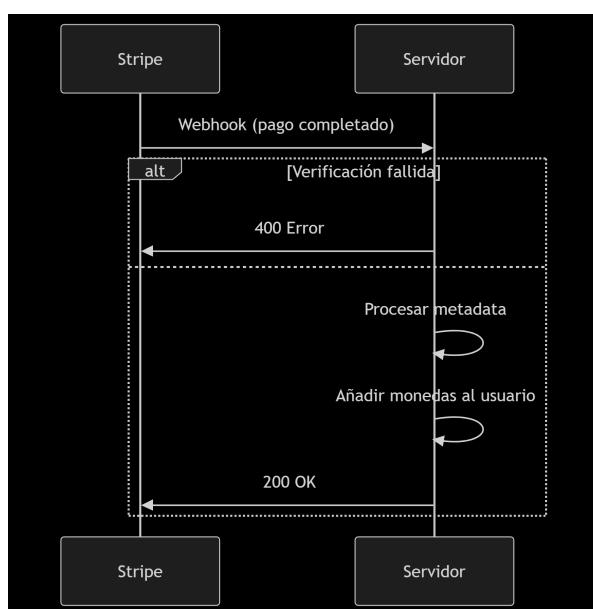
#### Ejemplos de Request JSON

Tabla: Json E/S Stripe webhook — manejo de notificaciones de pago

Json entrada	Json salida
{"data": { "object": { "metadata": { "userId": "usuario123", "coins": "100"{} } } }}	200 OK

#### Diagrama de flujo

Diagrama flujo: Stripe webhook — manejo de notificaciones de pago



Como vemos si nos llega una petición que no sea con una firma válida de mi cuenta Stripe no se completa el pago.

## 5.6 ReportesController

Este controlador usa la siguiente tabla:

Reporte:

Tabla: Reporte.

Campo	Tipo	Descripción
id	String	Id principal.
reportadorIdFirebase	String	Es el usuario que reporta
reportadoIdFirebase	String	Es el usuario que ha sido reportado

Este controlador usa las siguientes entidades:

UsuarioRegistroDTO:

Tabla: UsuarioRegistroDTO.

Campo	Tipo	Descripción
nombre	String	Nombre del usuario

### 5.6.1 /v1/reportes

Este controlador tiene los siguientes endpoint

Tabla: endpoints Rutina.

Método	Ruta	Descripción	Request Body	Response
POST	/reportar/{idFirebase}	Permite a un usuario autenticado reportar a otro usuario.	-	String

## 5.6.2 Reportar usuario

### Descripción funcional

Este endpoint POST /reportar/{idFirebase} permite a un usuario autenticado reportar a otro usuario. Cada usuario solo puede reportar una vez al mismo usuario. El sistema incrementa un contador de reportes en el perfil del usuario reportado.

### Validaciones

**idFirebase (usuario reportado):** Debe existir en la base de datos. Si no existe, se lanza NotFoundException con el mensaje "el usuario no existe".

**Usuario autenticado (reportador):** Se extrae del objeto Authentication.

**Duplicidad:** Si el usuario ya ha reportado antes al mismo usuario, se lanza ConflictException con el mensaje "Ya reportaste a este usuario."

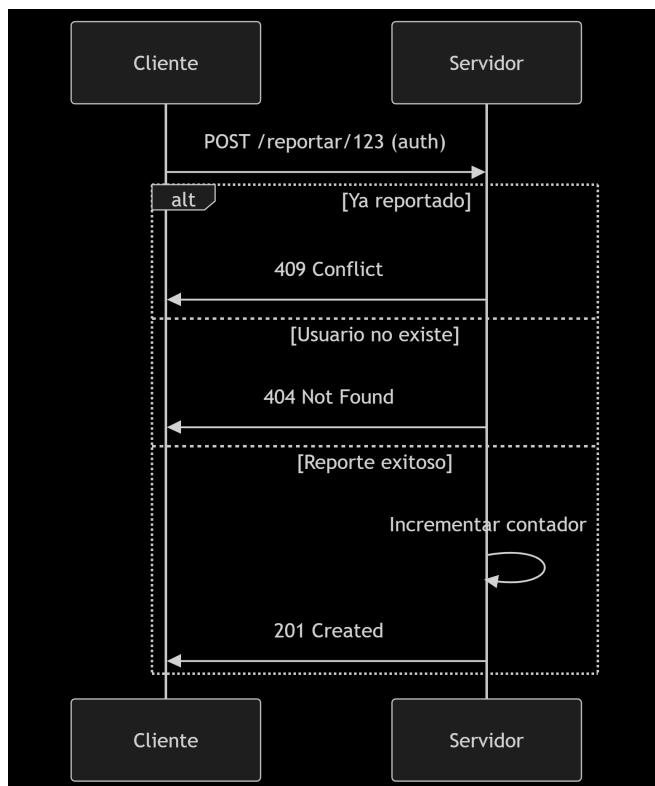
### Ejemplos de Request JSON

Tabla: Json E/S Reportar usuario.

Json entrada	Json salida
usuario456	"Reporte enviado correctamente."

### Diagrama de flujo

Diagrama flujo: Reportar usuario.



Como vemos, el usuario solo puede reportar a un usuario una sola vez.

## 5.7 ModeracionController

Este controlador se encarga de gestionar comentarios, eliminarlos o aprobarlos, también puede borrar a usuarios con reportes

Este controlador usa la siguiente tabla:

Rutina:

### 5.7.1 /v1/moderacion

Este controlador tiene los siguientes endpoint

Tabla: endpoints moderacion.

Método	Ruta	Descripción	Request Body	Response
GET	/verificar	Permite al administrador verificar comentarios con imágenes, que no le carga al público	-	List<ComentarioDto>
DELETE	/eliminar/{id}	Permite al administrador eliminar comentarios, que sean con imágenes explícitas, suma +1 en reporte al comentario del autor	-	-
GET	/reportados	Permite al administrador, obtener a los usuarios que tenga más de 1 reporte	-	-
Delete	/usuario/{id}	Permite al administrador, eliminar a los usuarios que tengan más de un reporte	-	-

### 5.7.2 Verificar comentarios pendientes de moderación

#### Descripción funcional

Este endpoint GET /verificar permite a un usuario con rol de administrador obtener una lista de comentarios que están pendientes de moderación (comentarios con estado false). Es utilizado para revisar y aprobar o rechazar comentarios con imágenes que puedan incumplir las normas de la comunidad.

#### Validaciones

**Autenticación:** El usuario debe estar autenticado.

**Permiso:** Solo usuarios con rol administrativo pueden acceder a esta lista. Si el usuario no es administrador, se lanza UnauthorizedException con el mensaje "No tienes permiso".

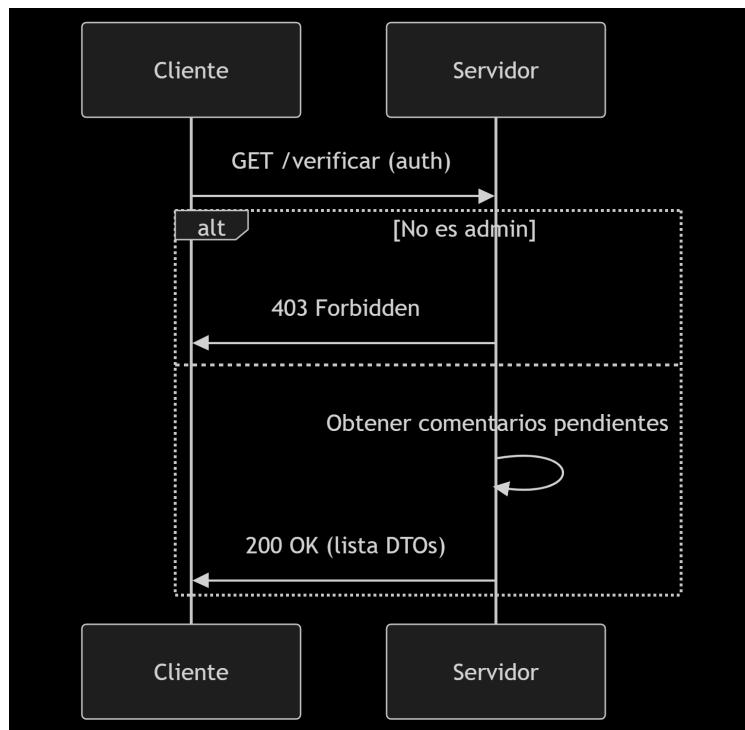
#### Ejemplos de Request JSON

Tabla: Json E/S verificar comentarios pendientes de moderación

Json entrada	Json salida
-	[{"id": "comentario123", "autorId": "usuario456", "texto": "Comentario imagen sospechosa", "estado": false, "fecha": "2025-06-09T12:00:00Z", "imagenUrl": "https://eple.com/ien.jpg"}]

#### Diagrama de flujo

Diagrama flujo: verificar comentarios pendientes de moderación.



Como vemos solo devuelve los comentarios con imágenes, ya sean explícitas o no

### 5.7.3 Eliminar comentario por administrador

#### Descripción funcional

Este endpoint `DELETE /eliminar/{id}` permite a un usuario con rol administrador eliminar un comentario específico identificado por su id. Esta acción se utiliza para retirar contenido que incumple las normas de la comunidad.

#### Validaciones

**Autenticación:** El usuario debe estar autenticado.

**Permiso:** Solo usuarios con rol admin pueden eliminar comentarios. Si el usuario no es administrador, se lanza `UnauthorizedException` con el mensaje "No tienes permiso".

**Existencia:** El comentario debe existir; de lo contrario, se lanza una excepción en `obtenerComentarioPorId`.

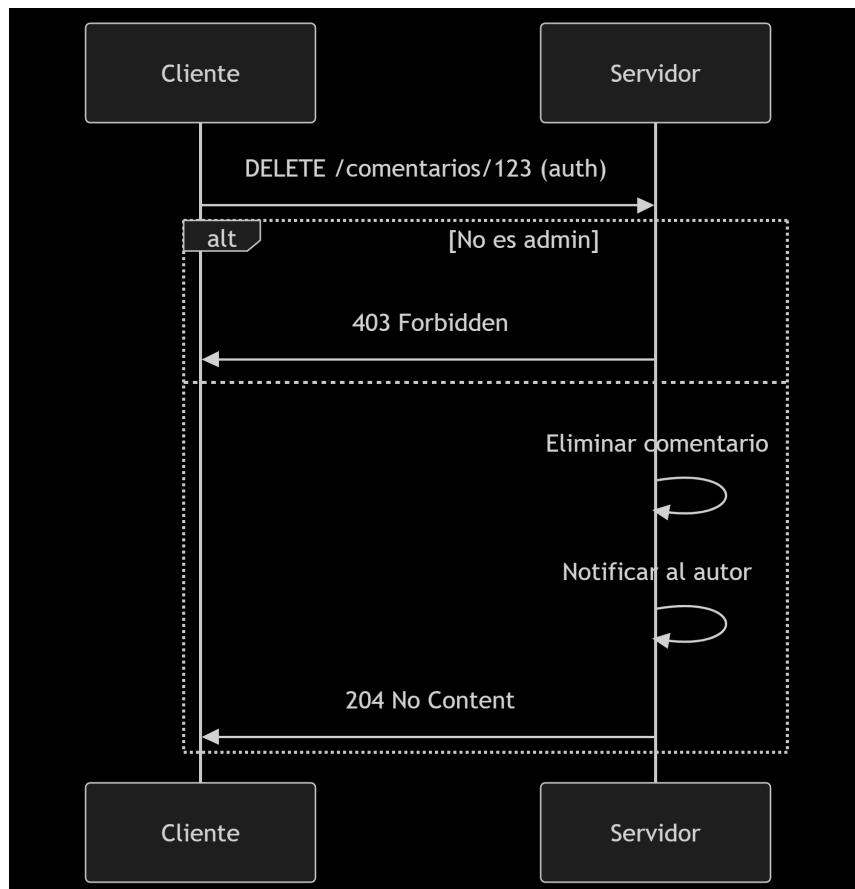
#### Ejemplos de Request JSON

Tabla: Json E/S eliminar comentario por administrador

Json entrada	Json salida
comentario123	204 No Content

#### Diagrama de flujo

Diagrama flujo: eliminar comentario por administrador.



Como vemos cuando se elimina al usuario, se le notifica que su cuenta ha sido eliminada.

#### 5.7.4 Obtener usuarios reportados

##### Descripción funcional

Este endpoint POST /registrarse permite a un nuevo usuario registrarse en la plataforma Rutify. La lógica de negocio incluye validaciones de datos, creación de usuario en Firebase Authentication, almacenamiento en Firestore y persistencia en MongoDB.

##### Validaciones

**Autenticación:** El usuario debe estar autenticado.

**Permiso:** Solo usuarios con rol admin pueden acceder a esta lista. Si el usuario no es admin, se lanza UnauthorizedException con el mensaje "No tienes permiso".

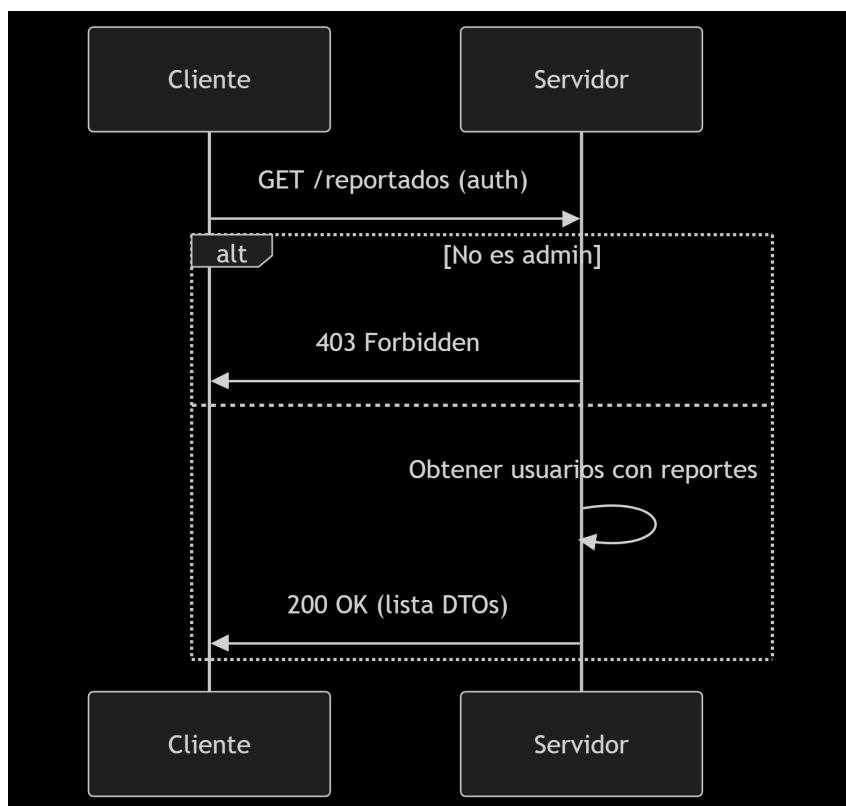
##### Ejemplos de Request JSON

Tabla: Json E/S obtener usuarios reportados

Json entrada	Json salida
-	[{"idFirebase": "usuario123", "nombre": "Juan Pérez", "sexo": "M", "esPremium": false, "avatar": "url_avatar_1"}]

##### Diagrama de flujo

Diagrama flujo: Obtener usuarios reportados.



Como vemos solo los admin pueden obtener la lista de reportados.

### 5.7.5 Eliminar usuario reportado

#### Descripción funcional

Este endpoint `DELETE /usuario/{id}` permite a un usuario con rol de administrador eliminar de forma permanente a otro usuario que ha sido reportado. Es parte del sistema de moderación para gestionar infracciones graves o reincidentes.

#### Validaciones

**Autenticación:** El usuario debe estar autenticado.

**Permiso:** Solo usuarios con rol admin pueden ejecutar esta acción. Si el usuario no es admin, se lanza `UnauthorizedException` con el mensaje "No tienes permiso".

**Usuario objetivo:** El id proporcionado debe corresponder a un usuario existente. Si no existe, se lanza una excepción desde `obtenerUsuario`.

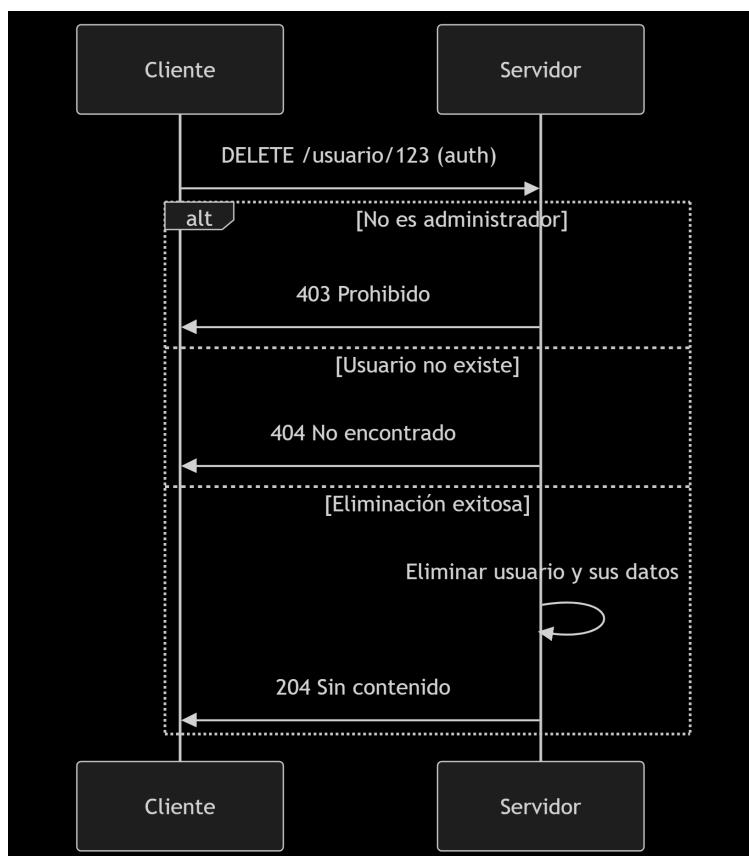
#### Ejemplos de Request JSON

Tabla: Json E/S Eliminar usuario reportado

Json entrada	Json salida
-	204 No Content

#### Diagrama de flujo

Diagrama flujo: Eliminar usuario reportado



Como vemos borramos toda la información del usuario al instante, y se le notifica por correo

## 5.8 EstadisticasDiariasController

Este controlador, se encarga de guardar las estadísticas de un día del usuario.

Este controlador usa la siguiente tabla:

EstadisticasDiarias:

Tabla: EstadisticasDiarias.

Campo	Tipo	Descripción
_id	String	Id principal.
idFirebase	String	Id del usuario asociado.
fecha	LocalDate	Fecha cuando se recogió esa estadística.
horasActivo	Double	Horas totales en las que el usuario, hizo ejercicio.
pesoCorporal	Double	Peso corporal del usuario ese día.
ejerciciosRealizados	Int	Cuántas repeticiones de ejercicios realizó en total.
kCaloriasQuemadas	Double	Total de KiloCalorías quemadas.

Este controlador usa las siguientes entidades:

EstadisticasDiariasDto:

Tabla: EstadisticasDiariasDto.

Campo	Tipo	Descripción
_id	String	Id principal.
idFirebase	String	Id del usuario asociado.
fecha	LocalDate	Fecha cuando se recogió esa estadística.
horasActivo	Double	Horas totales en las que el usuario, hizo ejercicio.
pesoCorporal	Double	Peso corporal del usuario ese día.
ejerciciosRealizados	Int	Cuántas repeticiones de ejercicios realizó en total.
kCaloriasQuemadas	Double	Total de KiloCalorías quemadas.

### 5.8.1 /v1/estadisticasDiarias

Este controlador tiene los siguientes endpoint

Tabla: endpoints Rutina.

Método	Ruta	Descripción	Request Body	Response
GET	/ultimosPesos	Proporciona los últimos 5 pesos del usuario, en caso de que no hay retorna una lista de 0.0	UsuarioRegistroDTO	List<Double>
PATCH	/v1/estadisticasDiarias	Permite al usuario actualizar las estadísticas ya creadas de ese día.	EstadisticasDiariasPatchDto	EstadisticasDiariasDto
GET	/dia	Permite al usuario obtener, una estadística diaria segun el dia	-	EstadisticasDiariasDto

### 5.8.2 Obtener últimos 5 pesos registrados

#### Descripción funcional

Este endpoint GET /ultimosPesos permite obtener los últimos 5 registros de peso corporal para un usuario específico, identificado por su id Firebase.

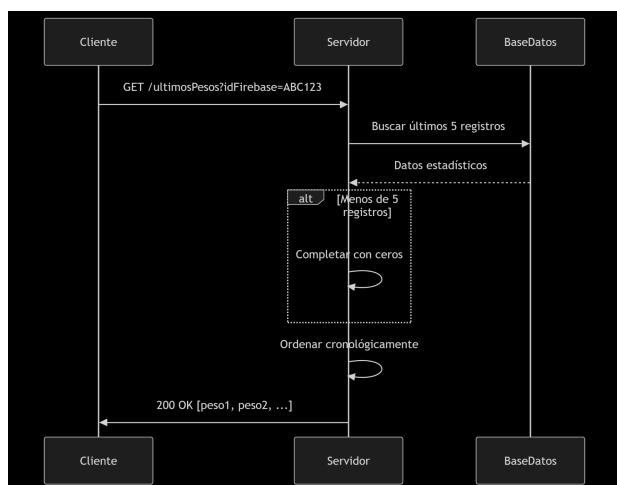
#### Ejemplos de Request JSON

Tabla: Json E/S Obtener últimos 5 pesos registrados

Json entrada	Json salida
/ultimosPesos?idFirebase=usuario123	[0.0, 0.0, 70.5, 70.8, 71.0]

#### Diagrama de flujo

Diagrama flujo: Obtener últimos 5 pesos registrados.



Como vemos, siempre nos retorna una lista de 5 elementos.

### 5.8.3 Actualizar estadísticas diarias

#### Descripción funcional

Este endpoint PATCH /estadisticasDiarias permite crear o actualizar las estadísticas diarias de un usuario identificado por su idFirebase y una fecha específica.

Si ya existen datos para ese día, se actualizan acumulando los valores numéricos. Si no existen, se crea un nuevo registro inicializando con valores predeterminados, y se toma el último peso corporal anterior como referencia si no se especifica uno nuevo.

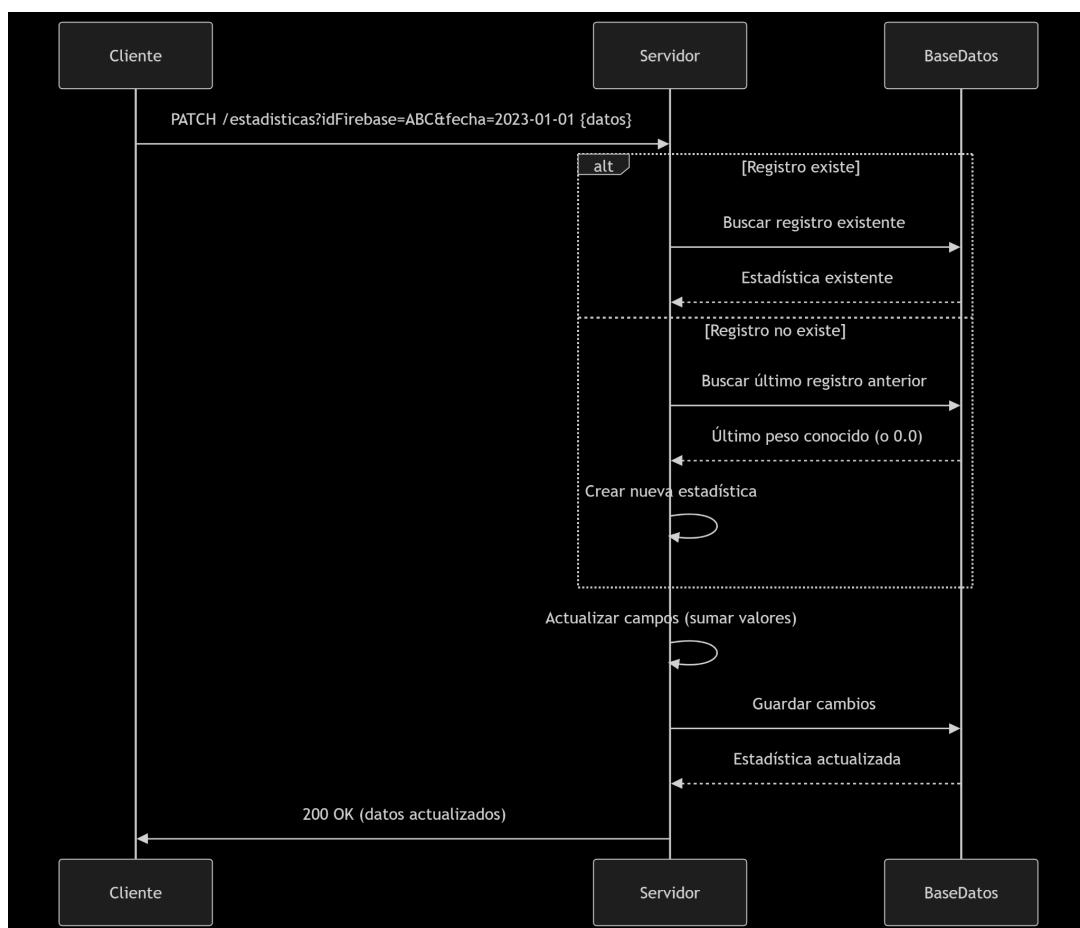
#### Ejemplos de Request JSON

Tabla: Json E/S Actualizar estadísticas diarias

Json entrada	Json salida
{ "horasActivo": 1.0, "kCaloriasQuemadas": 300.0 }	{"horasActivo": 1.5, "kCaloriasQuemadas": 300.0, "ejerciciosRealizados": 2, "pesoCorporal": 72.3}

#### Diagrama de flujo

Diagrama flujo: Actualizar estadísticas diarias.



Como vemos el flujo es unidireccional, y simple, o se crea uno nuevo o se actualiza el existente.

#### 5.8.4 Obtener estadísticas diarias de un día

##### Descripción funcional

Este endpoint GET /dia permite obtener las estadísticas diarias de un usuario específico (idFirebase) en una fecha concreta (fecha).

Si no existen estadísticas para ese día, se devuelve un error 404.

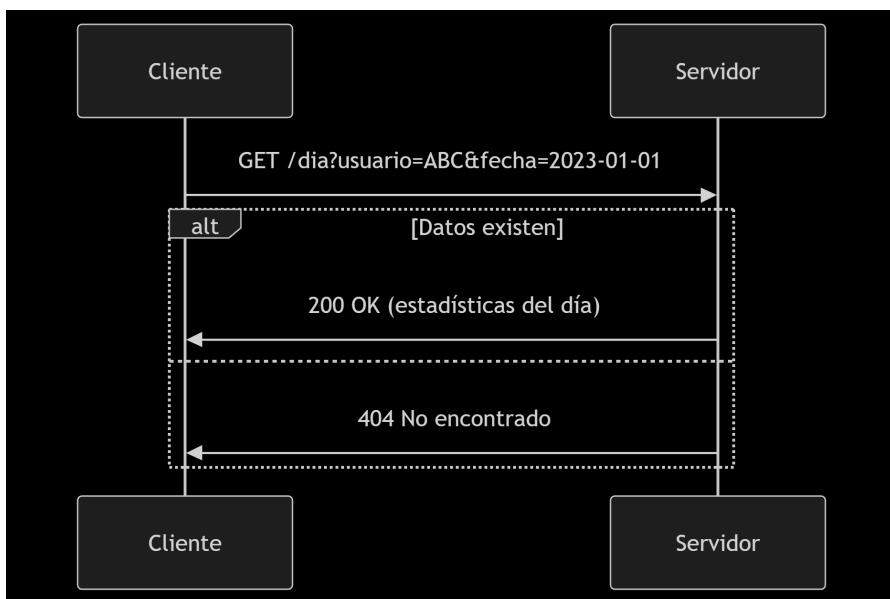
##### Ejemplos de Request JSON

Tabla: Json E/S Obtener estadísticas diarias de un día

Json entrada	Json salida
/dia?idFirebase=usuario123&fecha=2025-06-09	{"idFirebase": "usuario123", "fecha": "2025-06-09", "horasActivo": 2.5, "kCaloriasQuemadas": 500.0, "ejerciciosRealizados": 3, "pesoCorporal": 70.8}

##### Diagrama de flujo

Diagrama flujo: Obtener estadísticas diarias de un día.



Como vemos o devuelve la estadística o devuelve 404.

## 5.9 CoinPackController

Este controlador se encarga de suministrar packs de monedas virtuales, dentro de la app

Este controlador usa la siguiente tabla:

CoinPack:

Tabla: CoinPack.

Campo	Tipo	Descripción
id	String	Id principal.
nombre	String	Nombre del pack.
monedas	Int	Monedas que recibirá el usuario.
precio	Double	Cantidad que el usuario deberá abonar.
imageUrl	String	Url a una imagen referente al pack.

### 5.9.1 /v1/coin-packs

Este controlador tiene los siguientes endpoint

Tabla: endpoints coin-packs.

Método	Ruta	Descripción	Request Body	Response
GET	/v1/coin-packs	Obtiene todos los packs, vigentes a comprar	-	List<CoinPack>

### 5.9.2 usuario se registra

#### Descripción funcional

Este endpoint GET /packs permite obtener la lista completa de paquetes de monedas disponibles para su compra en la aplicación. Cada pack contiene una cantidad de monedas y su precio correspondiente.

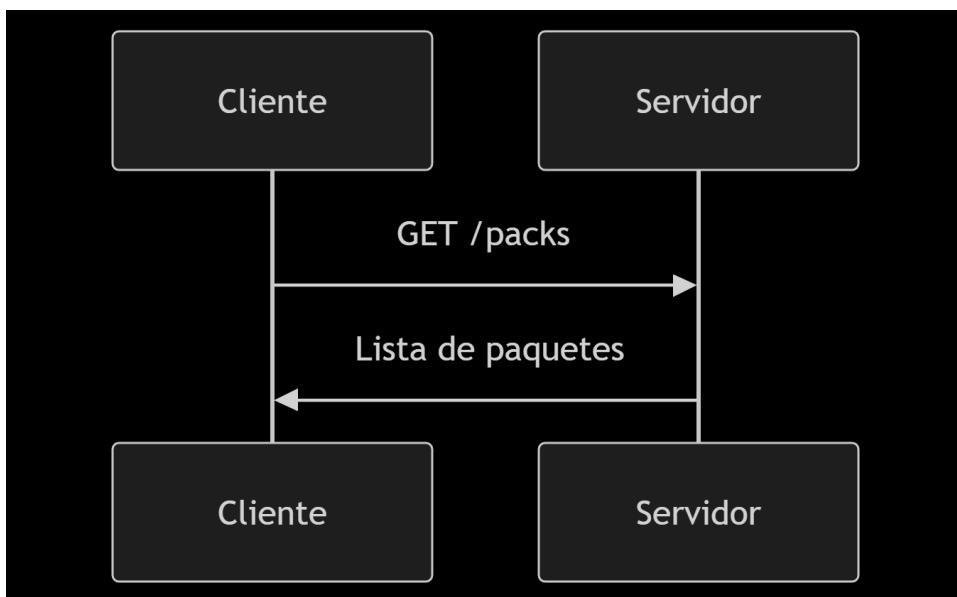
#### Ejemplos de Request JSON

Tabla: Json E/S usuario se registra

Json entrada	Json salida
-	[{ "id": "pack1", "cantidadMonedas": 100, "precio": 0.99 },]

#### Diagrama de flujo

Diagrama flujo: usuario se registra.



Como vemos, este endpoint da todos los packs existentes.

## 5.10 CosmeticoController

Este controlador se encarga de obtener los cosméticos, que puede comprar el usuario con monedas virtuales.

Este controlador usa la siguiente tabla:

Cosmético:

Tabla: Cosmetico.

Campo	Tipo	Descripción
_id	String	Id principal.
nombre	String	Nombre del cosmético.
tipo	String	Tipo de cosméticos, puede ser pantalón, brazos,camisa o tenis.
precioMonedas	Int	Coste en monedas virtuales.
imagenUrl	String	Url de la imagen del cosmético.

### 5.10.1 /v1/rutinas

Este controlador tiene los siguientes endpoint

Tabla: endpoints Cosmetico.

Método	Ruta	Descripción	Request Body	Response
GET	/v1/cosmeticos	Obtiene todos los cosméticos que el usuario puede comprar.	-	List<Cosmetico>

### 5.10.2 Obtener lista de cosméticos disponibles

#### Descripción funcional

Este endpoint GET /v1/cosmeticos permite obtener la lista completa de cosméticos disponibles en la aplicación. Los cosméticos pueden incluir elementos visuales como avatares, colores, fondos o accesorios, que personalizan la experiencia del usuario.

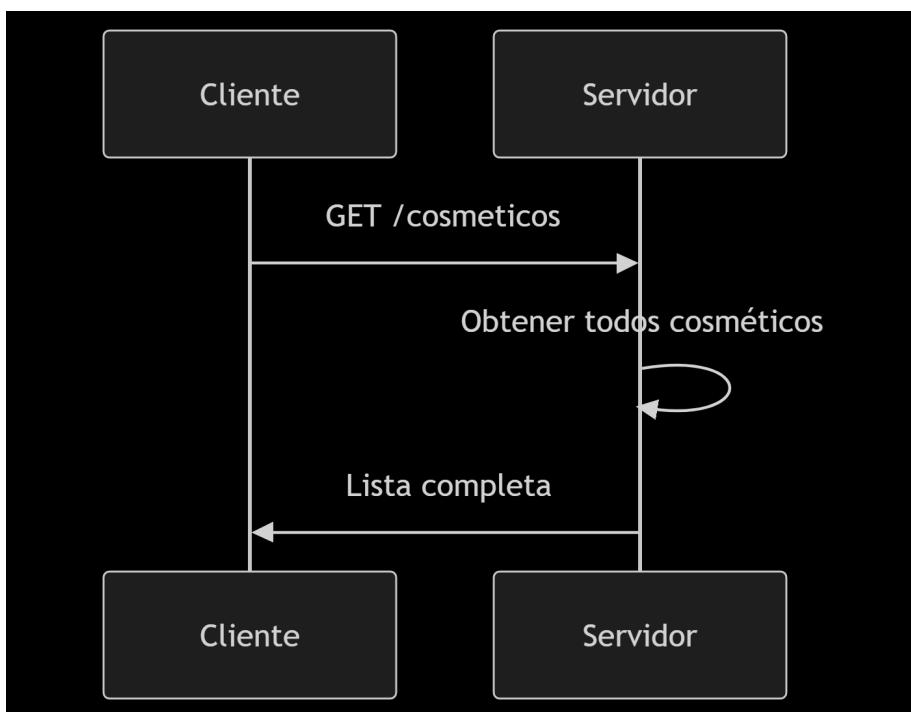
#### Ejemplos de Request JSON

Tabla: Json E/S Obtener lista de cosméticos disponibles

Json entrada	Json salida
/v1/cosmeticos	[{"id": "cosmetico1", "nombre": "Camisa de sol", "tipo": "Camisa", "precio": 100},]

#### Diagrama de flujo

Diagrama flujo: Obtener lista de cosméticos disponibles.



Como vemos, nos retorna todos los cosméticos en una sola consulta

## 5.11 ComprasController

Este Controlador se encarga de gestionar las compras y los productos obtenidos por el usuario.

Este controlador usa la siguiente tabla:

Compra:

Tabla: Compra.

Campo	Tipo	Descripción
id	String	Id principal que nos lo otorga Firebase Auth.
idUsuario	String	El id de usuario de Firebase que se asocia.
idCosmetico	String	El id de cosmético que ha comprado el usuario.
fechaCompra	LocalDateTime	El día que se efectuó la compra.

### 5.11.1 /v1/compras

Este controlador tiene los siguientes endpoint

Tabla: endpoints compras.

Método	Ruta	Descripción	Request Body	Response
GET	/{{idUsuario}}	Obtiene todas las compras que ha realizado el usuario, devuelve todos los cosméticos comprados.	-	List<Cosmetico>
POST	/v1/compras	Permite al usuario, hacer una compra y obtener un cosmético.	Compra	-

### 5.11.2 Obtener cosméticos comprados por el usuario

#### Descripción funcional

Este endpoint GET /v1/compras/{idUsuario} permite obtener la lista de todos los cosméticos que posee un usuario determinado. Esto incluye tanto los cosméticos comprados como los que vienen por defecto.

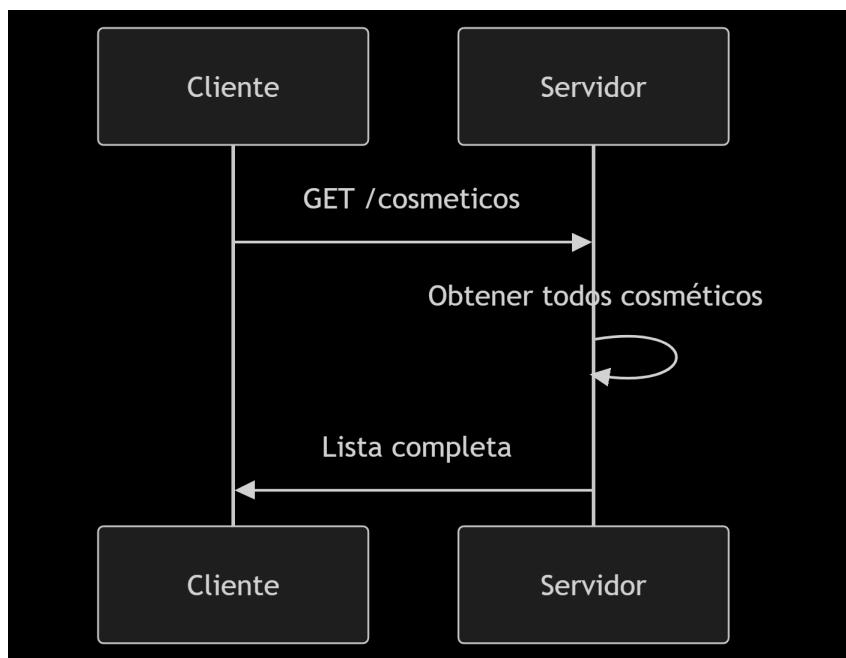
#### Ejemplos de Request JSON

Tabla: Json E/S Obtener cosméticos comprados por el usuario

Json entrada	Json salida
usuario123	[{"id": "cosmetico1", "nombre": "Gafas", "tipo": "accesorio", "precio": 100}, {"id": "cosmeticoBase", "nombre": "Avatar", "tipo": "avatar", "precio": 0}]

#### Diagrama de flujo

Diagrama flujo: Obtener cosméticos comprados por el usuario.



Como vemos en el diagrama, nos devuelve la lista de compras vigente por el usuario.

### 5.11.3 Registrar una compra de cosmético

#### Descripción funcional

Este endpoint POST /v1/compras permite a un usuario realizar la compra de un cosmético, siempre que no sea un cosmético que ya posea por defecto ni uno que ya haya comprado anteriormente. Se verifica que tenga suficiente saldo de monedas, y se descuenta el precio si la compra es válida.

#### Validaciones

**Cosmético por defecto:** El cosmético no debe estar en la lista idsDefecto.

**Compra duplicada:** Se comprueba si el usuario ya compró ese cosmético.

**Cosmético inexistente:** Se comprueba que el cosmético existe en la base de datos.

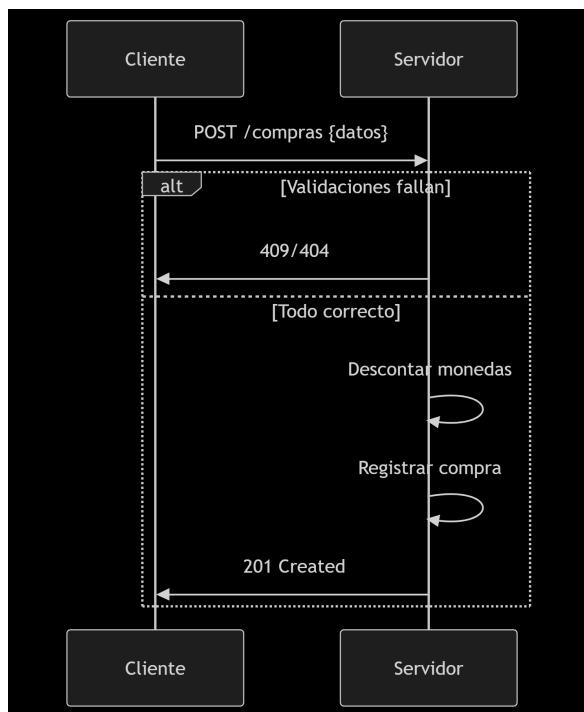
#### Ejemplos de Request JSON

Tabla: Json E/S Registrar una compra de cosmético

Json entrada	Json salida
{"id": "abc123", "idUsuario": "usuario001", "idCosmetico": "cosmetico045", "fechaCompra": "2025-06-09T14:30:00"}	"Compra registrada correctamente"

#### Diagrama de flujo

Diagrama flujo: Registrar una compra de cosmético.



Como vemos evitamos la inconsistencia de datos de comprar 2-3 veces el mismo producto.

## 5.12 EjerciciosController

Este controlador se encarga de proveer, agregar o modificar ejercicios para las rutinas.

Este controlador usa la siguiente tabla:

Ejercicio:

Tabla: Ejercicio.

Campo	Tipo	Descripción
id	String	Id principal.
nombreEjercicio	String	Nombre del ejercicio.
descripcion	String	Descripción del ejercicio.
imagen	String	Url de la imagen del ejercicio
equipo	String	Equipo necesario para realizar el ejercicio.
grupoMuscular	String	Grupo muscular al cual está enfocado el ejercicio.
caloriasQuemadasPorRepeticion	Double	Calorías que quemas por realizar una repetición.
puntoGanadosPorRepeticion	Double	Puntos de experiencia que gana el usuario por una sola repetición.

Este controlador usa las siguientes entidades:

EjercicioDTO:

Tabla: EjercicioDTO.

Campo	Tipo	Descripción
id	String	Id principal.
nombreEjercicio	String	Nombre del ejercicio.
descripcion	String	Descripción del ejercicio.
imagen	String	Url de la imagen del ejercicio
equipo	String	Equipo necesario para realizar el ejercicio.
grupoMuscular	String	Grupo muscular al cual está enfocado el ejercicio.
caloriasQuemadasPorRepeticion	Double	Calorías que quemas por realizar una repetición.
puntoGanadosPorRepeticion	Double	Puntos de experiencia que gana el usuario por una sola repetición.
Cantidad	Int	Número de repeticiones que deberá hacer el usuario.

### **5.12.1 /v1/ejercicios**

Este controlador tiene los siguientes endpoint

Tabla: endpoints ejercicios.

Método	Ruta	Descripción	Request Body	Response
POST	/registrarse	Permite al usuario crear un nuevo ejercicio.	EjercicioDTO	Ejercicio
GET	/retodiario	Permite al usuario, saber cual es el reto diario del día.	-	EjercicioDTO
GET	/obteneEjercicios	Permite al usuario obtener todos los ejercicios de la base de datos, para usarlo en las rutinas	-	List<EjercicioDTO>

### 5.12.2 Crear un ejercicio personalizado

#### Descripción funcional

Este endpoint POST /crear permite registrar un nuevo ejercicio en el sistema a partir de los datos proporcionados en el cuerpo de la solicitud. El ejercicio incluye información como el nombre, grupo muscular, equipo necesario, calorías quemadas y puntos ganados por repetición. Antes de guardar el ejercicio, se ejecuta una validación con la función validarEjercicio.

#### Validaciones

**Nombre del ejercicio:** no puede estar vacío.

**Descripción:** no puede estar vacía.

**Imagen:** no puede estar vacía y debe comenzar con http.

**Grupo muscular:** no puede estar vacío.

**Calorías por repetición:** debe ser un valor mayor que 0.

**Puntos por repetición:** debe ser un valor mayor que 0.

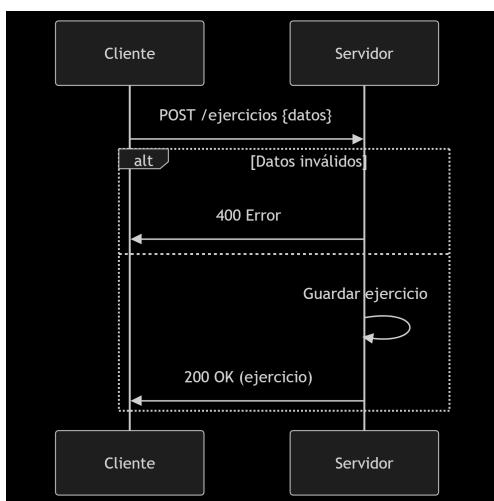
#### Ejemplos de Request JSON

Tabla: Json E/S Crear un ejercicio personalizado

Json entrada	Json salida
{"id": "ejercicio123", "nombreEjercicio": "Flexiones", "descripcion": "Ejercicio clásico para trabajar el pecho y tríceps", "imagen": "https://ruta-a-imagen.com/flexiones.jpg", "equipo": "Ninguno", "grupoMuscular": "Pecho", "caloriasQuemadasPorRepeticion": 0.45, "puntoGanadosPorRepeticion": 1.0}	{"id": "ejercicio123", "nombreEjercicio": "Flexiones", "descripcion": "Ejercicio clásico para trabajar el pecho y tríceps", "imagen": "https://ruta-a-imagen.com/flexiones.jpg", "equipo": "Ninguno", "grupoMuscular": "Pecho", "caloriasQuemadasPorRepeticion": 0.45, "puntoGanadosPorRepeticion": 1.0}

#### Diagrama de flujo

Diagrama flujo: Crear un ejercicio personalizado.



Siempre validamos los datos de los nuevos ejercicios, para que no haya inconsistencia.

### 5.12.3 Obtener reto diario

#### Descripción funcional

Este endpoint GET /v1/ejercicios/reto devuelve un único ejercicio aleatorio como reto diario. La selección del ejercicio se realiza de forma determinista en función de la fecha actual, por lo que todos los usuarios obtendrán el mismo reto cada día. Además, se asigna una cantidad aleatoria de repeticiones entre 20 y 50.

#### Validaciones

**Lista de ejercicios vacía:** Si no existen ejercicios registrados en la base de datos, se lanza una excepción NotFoundException.

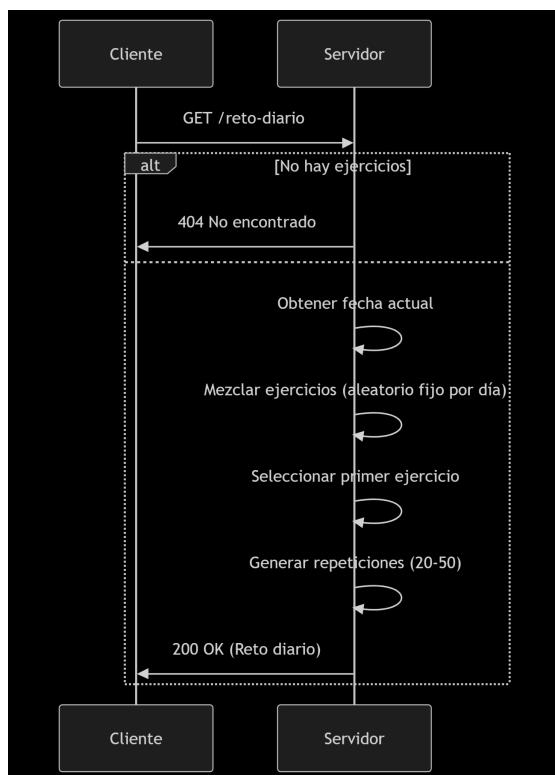
#### Ejemplos de Request JSON

Tabla: Json E/S Obtener reto diario

Json entrada	Json salida
-	{"id": "abc123", "nombreEjercicio": "Flexiones", "descripcion": "el pecho y tríceps", "equipo": "Nada", "grupoMuscular": "Pecho", "imagen": "https://ruta-a.com/flexiones.jpg", "caloriasQuemadasPorRepeticion": 0.45, "puntoGanadosPorRepeticion": 1.0, "cantidad": 37}

#### Diagrama de flujo

Diagrama flujo: Obtener reto diario.



Como vemos la implementación, es fija por dia, es decir un ejercicio y cantidad por dia completo

#### 5.12.4 Obtener lista de ejercicios

##### Descripción funcional

Este endpoint GET /v1/ejercicios/obteneEjercicios permite obtener una lista de ejercicios registrados. Se puede aplicar filtrado por grupo muscular y equipo, así como paginación mediante los parámetros page y size. Todos los filtros son opcionales.

##### Validaciones

Si no se proporcionan filtros ni paginación, se devuelven todos los ejercicios.

Los filtros aplican búsqueda con expresión regular insensible a mayúsculas (i).

La paginación se aplica solo si se pasa el parámetro page.

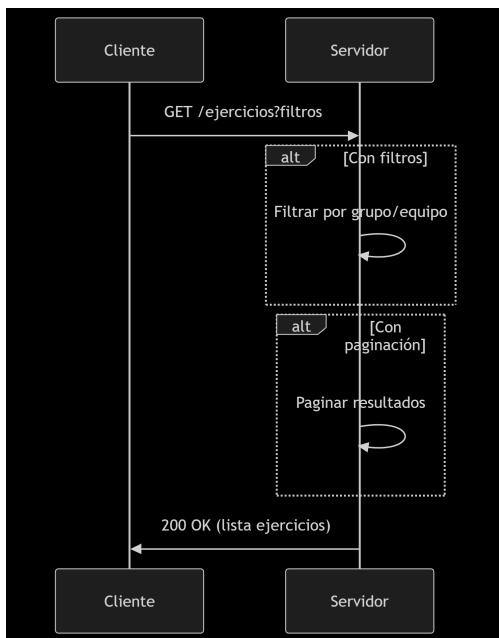
##### Ejemplos de Request JSON

Tabla: Json E/S Obtener lista de ejercicios

Json entrada	Json salida
-	[{"id": "abc123", "nombreEjercicio": "Flexiones", "descripcion": "Ejercicio para el tríceps", "equipo": "Ninguno", "grupoMuscular": "Pecho", "imagen": "https://ruta-.com/flexiones.jpg", "caloriasQuemadasPorRepeticion": 0.45, "puntoGanadosPorRepeticion": 1.0, "cantidad": 0}]

##### Diagrama de flujo

Diagrama flujo: Obtener lista de ejercicios.



Como vemos devuelve por paginación o filtros los diferentes ejercicios.

## 5.13 ComentariosController

Este controlador, gestiona y controla los diferentes comentarios en la aplicación Rutify.

Este controlador usa la siguiente tabla:

Comentario:

Tabla: Comentario.

Campo	Tipo	Descripción
_id	String	Id principal.
idFirebase	String	ID de firebase del usuario autor del comentario.
nombreUsuario	String	Nombre del autor del comentario.
avatarUrl	String	Url de la foto del usuario.
fechaPublicacion	LocalDate	Fecha donde se publicó el comentario.
estadoAnimo	String	Estado de ánimo que tiene el usuario.
texto	String	Texto del comentario.
imagenUrl	String	Url de la foto que contiene el comentario.
estado	Boolean	Estado del comentario, false necesita revisión por imagen no identificada, null foto revisada y comentario publicado.
idComentarioPadre	String	Si está null es un comentario principal, si tiene un ID es una respuesta a un comentario principal

Este controlador usa las siguientes entidades:

ComentarioDto:

Tabla: ComentarioDto.

Campo	Tipo	Descripción
_id	String	Id principal.
idFirebase	String	ID de firebase del usuario autor del comentario.
nombreUsuario	String	Nombre del autor del comentario.
avatarUrl	String	Url de la foto del usuario.
fechaPublicacion	LocalDate	Fecha donde se publicó el comentario.
estadoAnimo	String	Estado de ánimo que tiene el usuario.
texto	String	Texto del comentario.
imagenUrl	String	Url de la foto que contiene el comentario.
estado	Boolean	Estado del comentario, false necesita revisión por imagen no identificada, null foto revisada y comentario publicado.
idComentarioPadre	String	Si está null es un comentario principal, si tiene un ID es una respuesta a un comentario principal

### 5.13.1 /v1/comentarios

Este controlador tiene los siguientes endpoint

Tabla: endpoints comentarios.

Método	Ruta	Descripción	Request Body	Response
POST	/comentarios	Permite al usuario crear un comentario en Rutify.	ComentarioDto	ComentarioDto
GET	/comentarios	Permite al usuario visualizar los comentarios con estado null.	-	List<ComentarioDto>
GET	/comentarios/{id}/respuestas	Permite al usuario obtener respuestas de un comentario padre.	-	List<ComentarioDto>
POST	/comentarios/respuestas	Permite responder con un comentario, a un comentario padre.	ComentarioDto	ComentarioDto
DELETE	/comentarios/{idComentario}	Permite eliminar un comentario.	-	-
PUT	/comentarios/aprobar	Permite a los administradores aprobar un comentario con foto.	ComentarioDto	-
GET	/autor/{creadorId}	Permite al usuario obtener todos los comentarios de un usuario.	-	List<ComentarioDto>
GET	/autorComentario/{nombre}	Permite obtener, los comentario buscando por su nombre parcial	-	List<ComentarioDto>

### 5.13.3 Crear un comentario

#### Descripción funcional

Este endpoint POST /v1/comunidad/comentarios permite a un usuario crear un nuevo comentario en la comunidad, con opción de incluir una imagen.

Si se adjunta una imagen, el comentario queda en estado pendiente (estado = false).

Si no se adjunta imagen, el campo estado queda como nulo.

#### Validaciones

La parte "comentario" debe contener un objeto JSON válido con los campos requeridos.

La parte "imagen" es opcional. Si se incluye, debe ser un archivo válido.

El estado del comentario se asigna automáticamente según si tiene imagen o no.

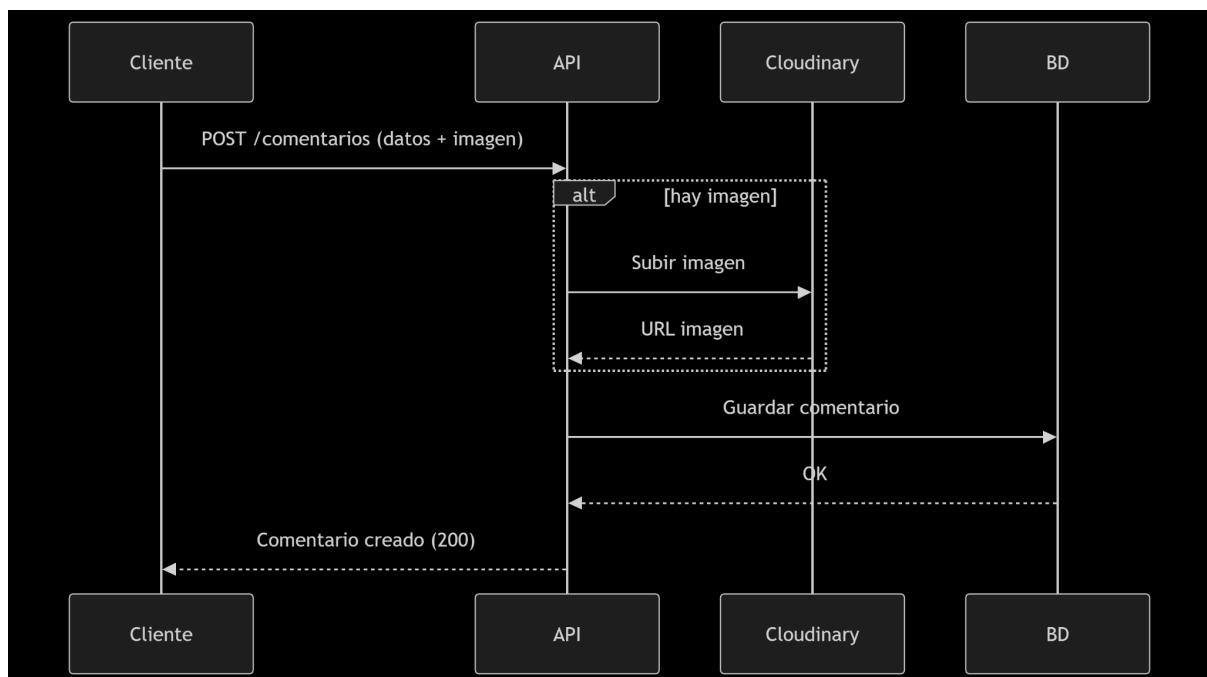
#### Ejemplos de Request JSON

Tabla: Json E/S Crear un comentario

Json entrada	Json salida
{"idFirebase": "user_001", "nombreUsuario": "Juan", "avatarUrl": "https://cdn..com/avatar.jpg", "fechaPublicacion": "2025-06-09T17:00:00", "estadoAnimo": "Feliz", "texto": "¡Mi primer día en el gym!"}	{"idFirebase": "user_001", "nombreUsuario": "Juan", "avatarUrl": "https://cdn..com/avatar.jpg", "fechaPublicacion": "2025-06-09T17:00:00", "estadoAnimo": "Feliz", "texto": "¡Mi primer día en el gym!"}

#### Diagrama de flujo

Diagrama flujo: Crear un comentario.



Como vemos los comentarios pueden tener o no imágenes, los comentarios que los contengan tiene que pasar una revisión

#### 5.13.4 Obtener comentarios principales públicos

##### Descripción funcional

Este endpoint GET /v1/comunidad/comentarios devuelve todos los comentarios principales que no son respuestas (idComentarioPadre = null) y cuyo estado es null (es decir, comentarios sin imagen o que no requieren aprobación).

Los comentarios se devuelven en orden inverso al de creación, mostrando los más recientes primero.

##### Validaciones

**idComentarioPadre IS NULL:** solo comentarios principales, no respuestas.

**Estado IS NULL:** comentarios públicos y ya visibles.

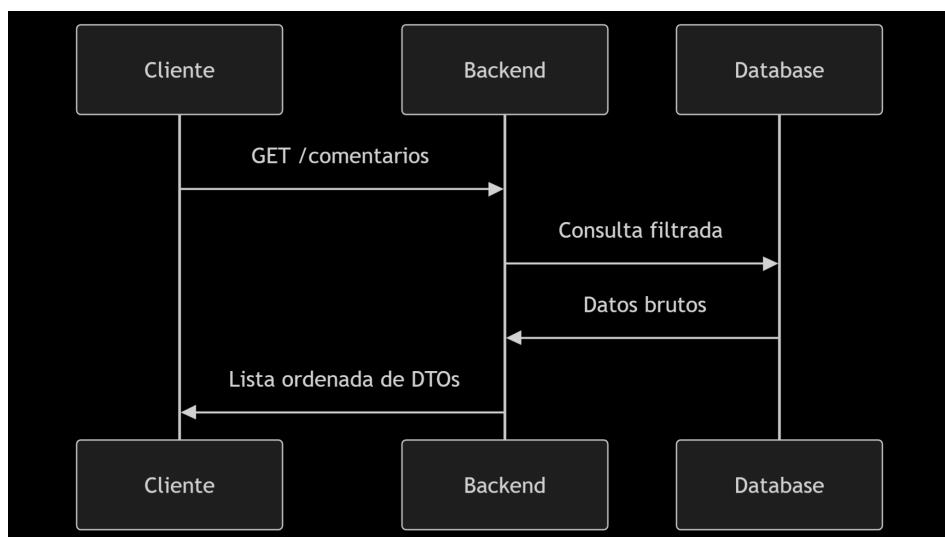
##### Ejemplos de Request JSON

Tabla: Json E/S Obtener comentarios principales públicos

Json entrada	Json salida
-	[{"id": "abc123", "idFirebase": "user_001", "nombreUsuario": "Juan", "avatarUrl": "https://cdn.ejemplo.com/avatar.jpg", "fechaPublicacion": "2025-06-09T17:00:00", "imagenUrl": null, "estadoAnimo": "Feliz", "texto": "¡Mi primer día en el gym!", "idComentarioPadre": null, "estado": null}]

##### Diagrama de flujo

Diagrama flujo: Obtener comentarios principales públicos.



Como vemos solo aparecen los comentarios que pasaron dicha revisión.

### 5.13.5 Obtener comentario principal y sus respuestas

#### Descripción funcional

Este endpoint GET /v1/comunidad/comentarios/{id}/respuestas devuelve un comentario principal identificado por su id junto con todas sus respuestas directas (comentarios hijos).

El primer objeto del array corresponde siempre al comentario padre, seguido de sus respuestas.

#### Validaciones

Si no existe un comentario con el ID indicado, se lanza una excepción 404 con el mensaje: "este comentario no existe".

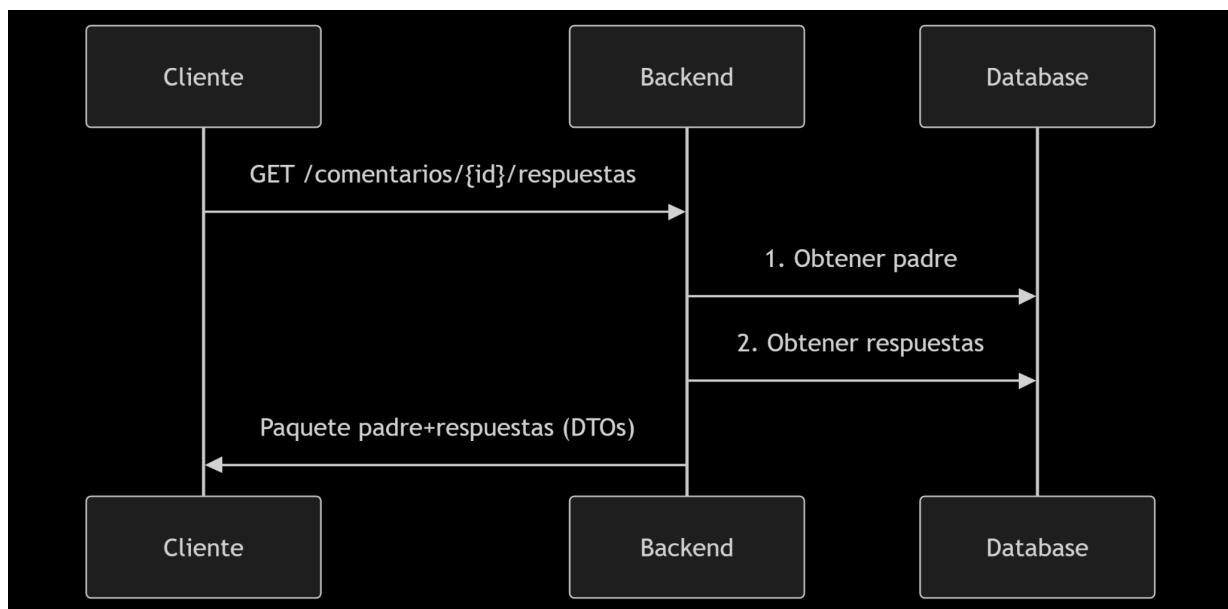
#### Ejemplos de Request JSON

Tabla: Obtener comentario principal y sus respuestas

Json entrada	Json salida
/v1/comunidad/comentarios/123/respuestas	[{"id": "abc123", "idFirebase": "user_001", ...},...]

#### Diagrama de flujo

Diagrama flujo: Obtener comentario principal y sus respuestas



Como vemos, nos devuelve todos los comentarios relacionados con el padre

### 5.13.6 Crear respuesta a un comentario

#### Descripción funcional

Este endpoint POST /v1/comunidad/comentarios/resuestas permite a un usuario crear una respuesta a un comentario existente. La respuesta se envía como un objeto ComentarioDto en el cuerpo de la petición. El campo idComentarioPadre debe contener el ID del comentario al que se está respondiendo.

#### Validaciones

El campo idComentarioPadre debe estar presente en el ComentarioDto, indicando el comentario padre al que se responde.

No se permite crear respuestas sin referencia a un comentario padre.

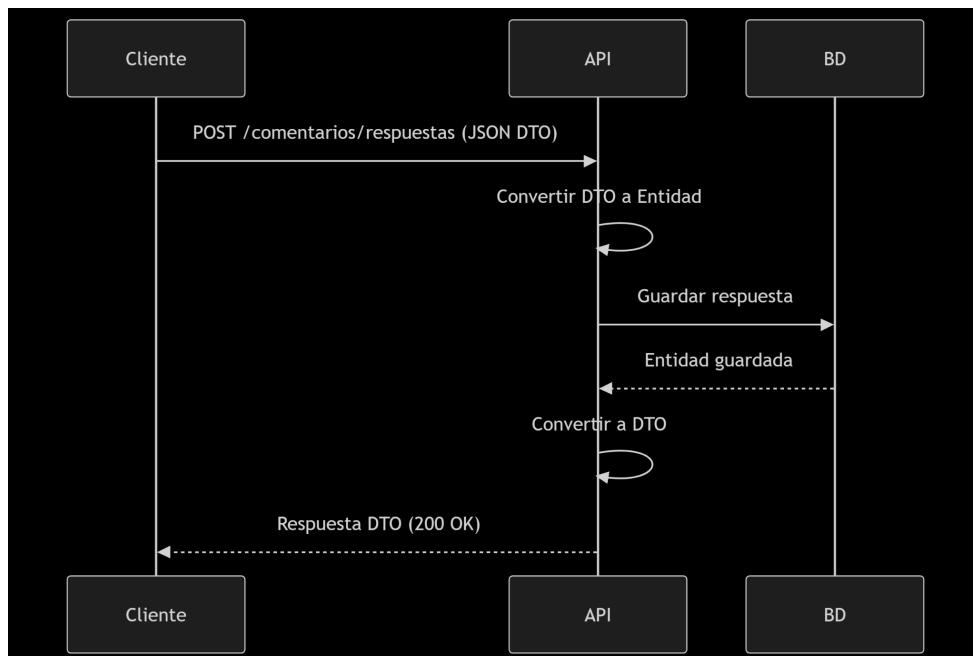
#### Ejemplos de Request JSON

Tabla: Json E/S Crear respuesta a un comentario

Json entrada	Json salida
{"idFirebase": "user_002", "nombreUsuario": "María", "avatarUrl": "https://cdn..com/avatar2.jpg", "fechaPublicacion": "2025-06-09T18:00:00", "imagenUrl": null, "estadoAnimo": "Animada", "texto": "¡Ánimo! Ya verás los resultados 😊", "idComentarioPadre": "abc123", "estado": null}	{"idFirebase": "user_002", "nombreUsuario": "María", "avatarUrl": "https://cdn..com/avatar2.jpg", "fechaPublicacion": "2025-06-09T18:00:00", "imagenUrl": null, "estadoAnimo": "Animada", "texto": "¡Ánimo! Ya verás los resultados 😊", "idComentarioPadre": "abc123", "estado": null}

#### Diagrama de flujo

Diagrama flujo: Crear respuesta a un comentario.



Como vemos, la api nos guarda el comentario y nos lo devuelve

### 5.13.7 Eliminar comentario o respuesta

#### Descripción funcional

Este endpoint `DELETE /v1/comunidad/comentarios/{idComentario}` permite a un usuario eliminar un comentario o una respuesta. Solo puede eliminarlo si:

Es el autor del comentario.

Es el autor del comentario padre al que pertenece la respuesta.

Tiene rol de administrador.

Además, si el comentario contiene una imagen alojada en Cloudinary, esta también se elimina automáticamente. Si se trata de un comentario principal, también se eliminan todas sus respuestas.

#### Validaciones

El comentario debe existir.

El usuario debe ser el autor del comentario ó del padre, o tener rol admin.

Si hay imagen en Cloudinary, se elimina con `cloudinary.uploader().destroy(...)`.

Si es un comentario padre, se eliminan primero sus respuestas (por `deleteByIdComentarioPadreEquals`)

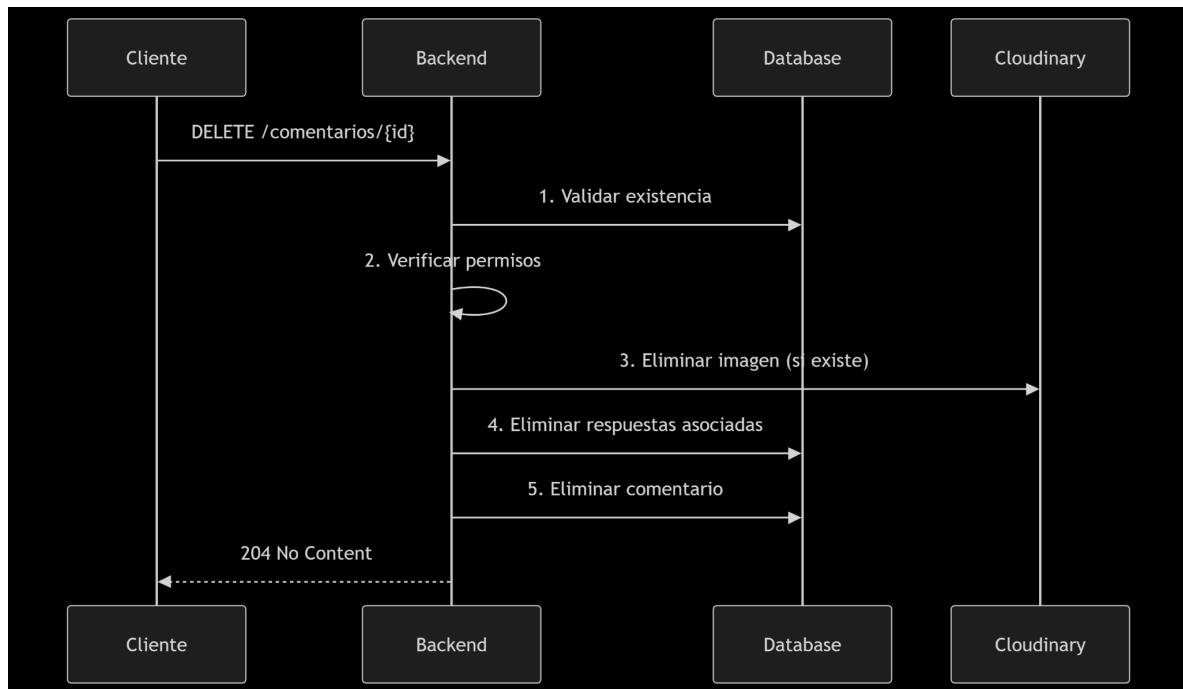
#### Ejemplos de Request JSON

Tabla: Json E/S Eliminar comentario o respuesta

Json entrada	Json salida
<code>idComentario</code>	HTTP 204 No Content

#### Diagrama de flujo

Diagrama flujo: Eliminar comentario o respuesta.



Se elimina todo, no dejamos nada guardado fotos, respuestas y comentarios.

### 5.13.8 Aprobar comentario

#### Descripción funcional

Este endpoint PUT /v1/comunidad/comentarios/aprobar permite a un usuario con rol administrativo aprobar un comentario que estaba pendiente de revisión (es decir, que tenía el campo estado distinto de null). La aprobación consiste en cambiar el campo estado a null para que el comentario sea público y visible.

#### Validaciones

Solo usuarios con rol administrativo pueden aprobar comentarios.

El comentario debe existir.

Cambiar estado a null en la base de datos.

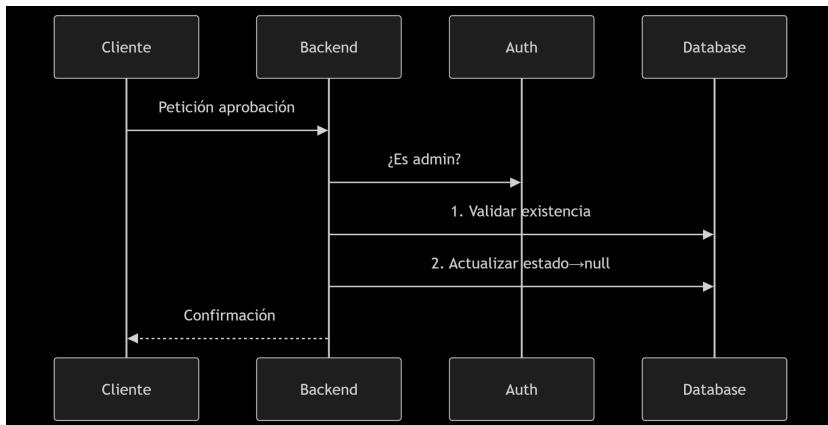
#### Ejemplos de Request JSON

Tabla: Json E/S Aprobar comentario

Json entrada	Json salida
{ "_id": "abc123", "idFirebase": "user_001", "nombreUsuario": "Juan", "avatarUrl": "https://cdn.ejemplo.com/avatar.jpg", "fechaPublicacion": "2025-06-09T17:00:00", "imagenUrl": "https://res.cloudinary.com/.../imagen.jpg", "estadoAnimo": "Feliz", "texto": "Comentario con imagen para aprobación", "idComentarioPadre": null, "estado": false}...]	-

#### Diagrama de flujo

Diagrama flujo: Aprobar comentario.



Como vemos, tenemos validación por roles, y el administrador es el que decide.

### 5.13.9 Obtener comentarios por autor

#### Descripción funcional

Este endpoint GET /v1/comunidad/comentarios/autor/{creadorId} permite obtener la lista de comentarios principales (no respuestas) realizados por un autor específico, identificado por su idFirebase.

#### Validaciones

El creador no puede estar vacío o nulo.

Solo devuelve comentarios que sean comentarios principales (sin padre).

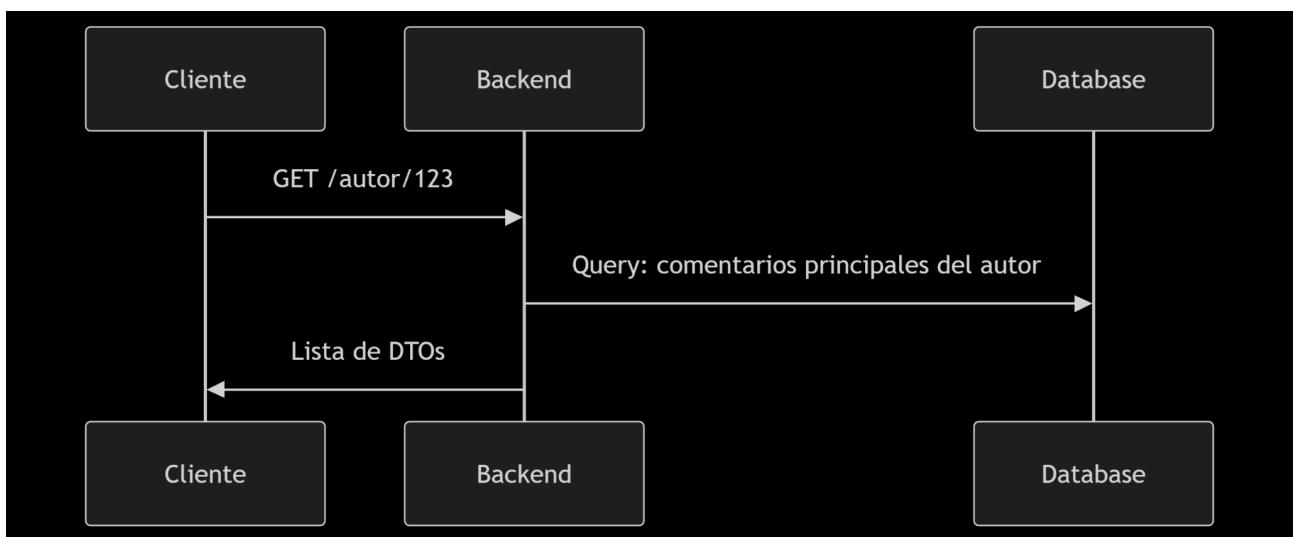
#### Ejemplos de Request JSON

Tabla: Json E/S Obtener comentarios por autor

Json entrada	Json salida
/v1/comunidad/comentarios/autor/user_001	[ { "_id": "abc123", "idFirebase": "user_001", "nombreUsuario": "Juan", "avatarUrl": "https://cdn.ejemplo.com/avatar.jpg", "fechaPublicacion": "2025-06-09T17:00:00", "imagenUrl": "https://res.cloudinary.com/.../imagen.jpg", "estadoAnimo": "Feliz", "texto": "Comentario principal de Juan", "idComentarioPadre": null, "estado": null...] ]

#### Diagrama de flujo

Diagrama flujo: Obtener comentarios por autor.



Devuelve los comentarios que están verificados y del mismo autor.

### 5.13.10 Obtener comentarios por nombre de autor

#### Descripción funcional

Este endpoint GET /v1/comunidad/comentarios/autorComentario/{nombre} permite obtener una lista de comentarios (tanto principales como respuestas) en los que el campo nombreUsuario contenga el nombre indicado, sin diferenciar mayúsculas de minúsculas.

#### Validaciones

El nombre no puede estar vacío.

La búsqueda es insensible a mayúsculas y minúsculas (IgnoreCase).

Se devuelven todos los comentarios coincidentes, sin filtrar si son principales o respuestas.

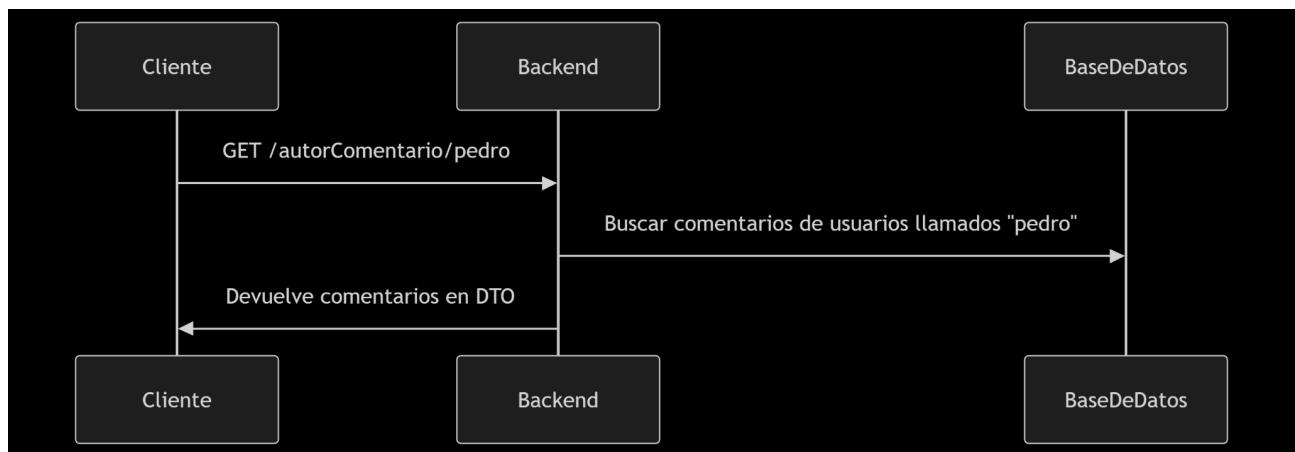
#### Ejemplos de Request JSON

Tabla: Json E/S Obtener comentarios por nombre de autor

Json entrada	Json salida
/v1/comunidad/comentarios/autorComentario/juan	[{"_id": "abc123", "idFirebase": "user_001", "nombreUsuario": "Juan Pérez", "avatarUrl": " <a href="https://cdn..com/avatar.jpg">https://cdn..com/avatar.jpg</a> ", ...}]

#### Diagrama de flujo

Diagrama flujo: Obtener comentarios por nombre de autor.



Como vemos, solo retornamos los comentarios que se parezca al nombre parcial de autor.

## 5.14 EstadisticasController

Este controlador gestiona las estadísticas que ha conseguido el usuario, en nuestra app

Este controlador usa la siguiente tabla:

Estadísticas:

Tabla: Estadísticas.

Campo	Tipo	Descripción
_id	String	Id principal.
idUserbase	String	Id de usuario al que está asociado.
lvlBrazo	Double	Nivel de fuerza del brazo.
lvlPecho	Double	Nivel de fuerza del pecho.
lvlAbdominal	Double	Nivel de fuerza de los abdominales.
pesoCorporal	Double	Peso corporal del usuario.
lvlEspalda	Double	Nivel de fuerza de la espalda.
lvlPiernas	Double	Nivel de fuerza de las piernas.
horasActivo	Double	Horas de entrenamiento totales del usuario.
ejerciciosRealizados	Double	Cantidad de los ejercicios totales realizados
kCaloriasQuemadas	Double	Cantidad de Kilocalorías quemadas totales

Este controlador usa las siguientes entidades:

EstadisticasDto:

Tabla: EstadisticasDto.

Campo	Tipo	Descripción
_id	String	Id principal.
idUser	String	Id de usuario al que está asociado.
lvlBrazo	Double	Nivel de fuerza del brazo.
lvlPecho	Double	Nivel de fuerza del pecho.
lvlAbdominal	Double	Nivel de fuerza de los abdominales.
pesoCorporal	Double	Peso corporal del usuario.
lvlEspalda	Double	Nivel de fuerza de la espalda.
lvlPiernas	Double	Nivel de fuerza de las piernas.
horasActivo	Double	Horas de entrenamiento totales del usuario.
ejerciciosRealizados	Double	Cantidad de los ejercicios totales realizados
kCaloriasQuemadas	Double	Cantidad de Kilocalorías quemadas totales

#### **5.14.1 /v1/rutinas**

Este controlador tiene los siguientes endpoint

Tabla: endpoints Rutina.

Método	Ruta	Descripción	Request Body	Response
POST	/crear	Permite al usuario crear su estadística en Rutify.	Estadisticas	EstadisticasDto
GET	/{usuarioid}	Permite ver al usuario sus estadísticas	-	EstadisticasDto
PATCH	/{usuarioid}	Permite al usuario actualizar su estadísticas ya existentes.	EstadisticasPatchDto	EstadisticasDto

### 5.14.2 Obtener estadísticas de usuario

#### Descripción funcional

Este endpoint GET /v1/comunidad/estadisticas/{usuarioid} permite obtener las estadísticas personales asociadas a un usuario específico, utilizando su idFirebase.

#### Validaciones

El ID del usuario (usuarioid) debe existir.

Si no se encuentran estadísticas asociadas a ese ID, se lanza un error 404 Not Found.

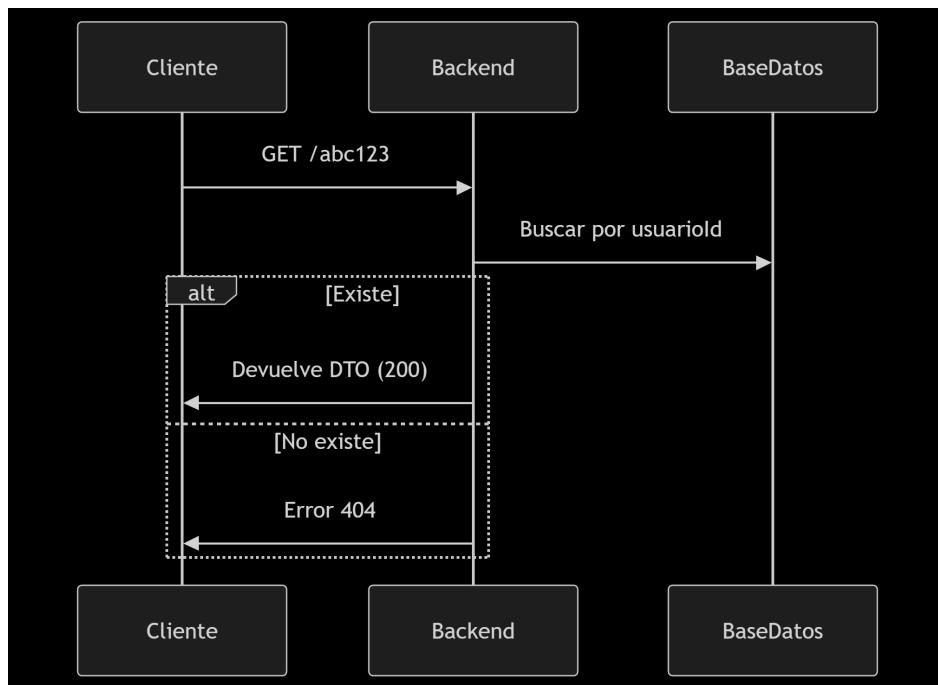
#### Ejemplos de Request JSON

Tabla: Json E/S Obtener estadísticas de usuario

Json entrada	Json salida
/v1/comunidad/estadisticas/user_001	{"idFirebase": "user_001", "lvlPiernas": 3, "lvlBrazos": 2, "lvlPecho": 1, "lvlEspalda": 4}

#### Diagrama de flujo

Diagrama flujo: Obtener estadísticas de usuario.



Como vemos el flujo de trabajo es sencillo, o existen las estadísticas o retorna un 404

### 5.14.3 Crear estadísticas de usuario

#### Descripción funcional

Este endpoint POST /v1/comunidad/estadisticas/crear permite a un usuario crear por primera vez su conjunto de estadísticas personales.

Cada usuario solo puede tener un único documento de estadísticas, identificado por su idFirebase.

#### Validaciones

El usuario debe estar autenticado.

Solo se permite una estadística por usuario. Si ya existe una, se lanza un error 409 Conflict.

Se guardan los datos proporcionados en la base de datos.

El resultado se devuelve como un EstadisticasDto.

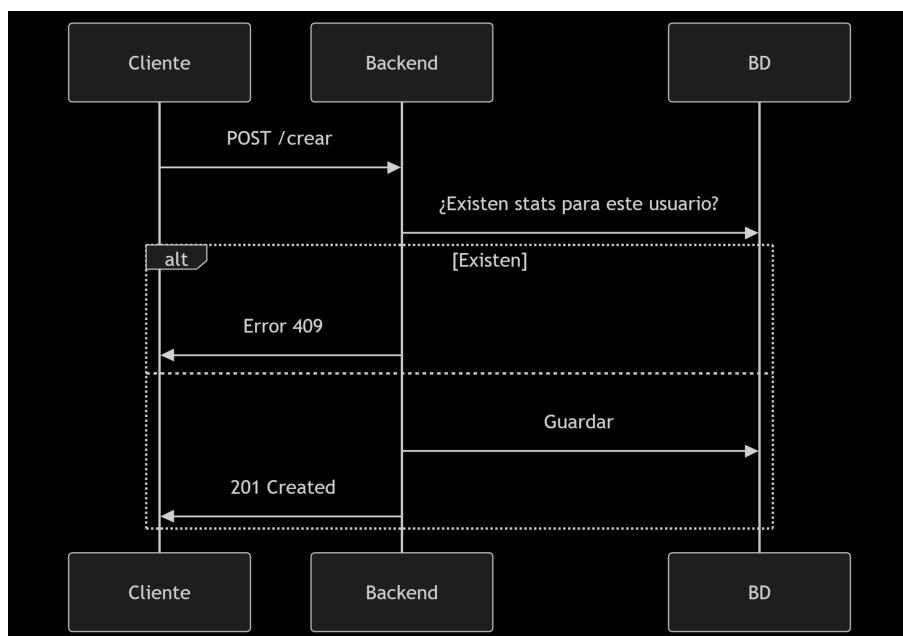
#### Ejemplos de Request JSON

Tabla: Json E/S Crear estadísticas de usuario

Json entrada	Json salida
{ "idFirebase": "user_001", "lvlPiernas": 3, "lvlBrazos": 2, "lvlPecho": 1, "lvlEspalda": 4 }	{ "idFirebase": "user_001", "lvlPiernas": 3, "lvlBrazos": 2, "lvlPecho": 1, "lvlEspalda": 4 }

#### Diagrama de flujo

Diagrama flujo: Crear estadísticas de usuario.



Como vemos un usuario solo puede tener una estadísticas asociadas a él.

#### 5.14.4 Actualizar estadísticas de usuario

##### Descripción funcional

Este endpoint PATCH /v1/comunidad/estadisticas/{usuarioid} permite a un usuario actualizar parcialmente sus estadísticas personales. Solo el propio usuario puede modificar sus datos.

##### Validaciones

El usuarioid debe coincidir con el ID del usuario autenticado.

Las estadísticas deben existir previamente.

Solo se actualizan los campos que estén presentes (no nulos) en el cuerpo del request.

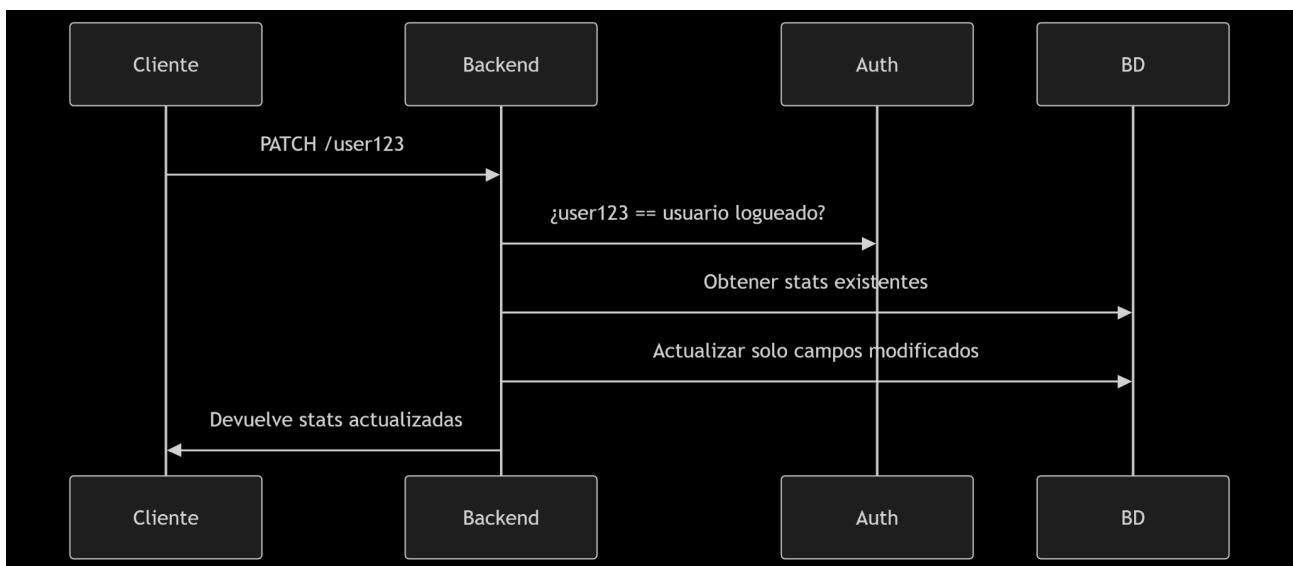
##### Ejemplos de Request JSON

Tabla: Json E/S Actualizar estadísticas de usuario

Json entrada	Json salida
{"lvlBrazo": 3, "kCaloriasQuemadas": 1200}	{"idFirebase": "user_001", "lvlPiernas": 2, "lvlBrazo": 3,...}

##### Diagrama de flujo

Diagrama flujo: Actualizar estadísticas de usuario.

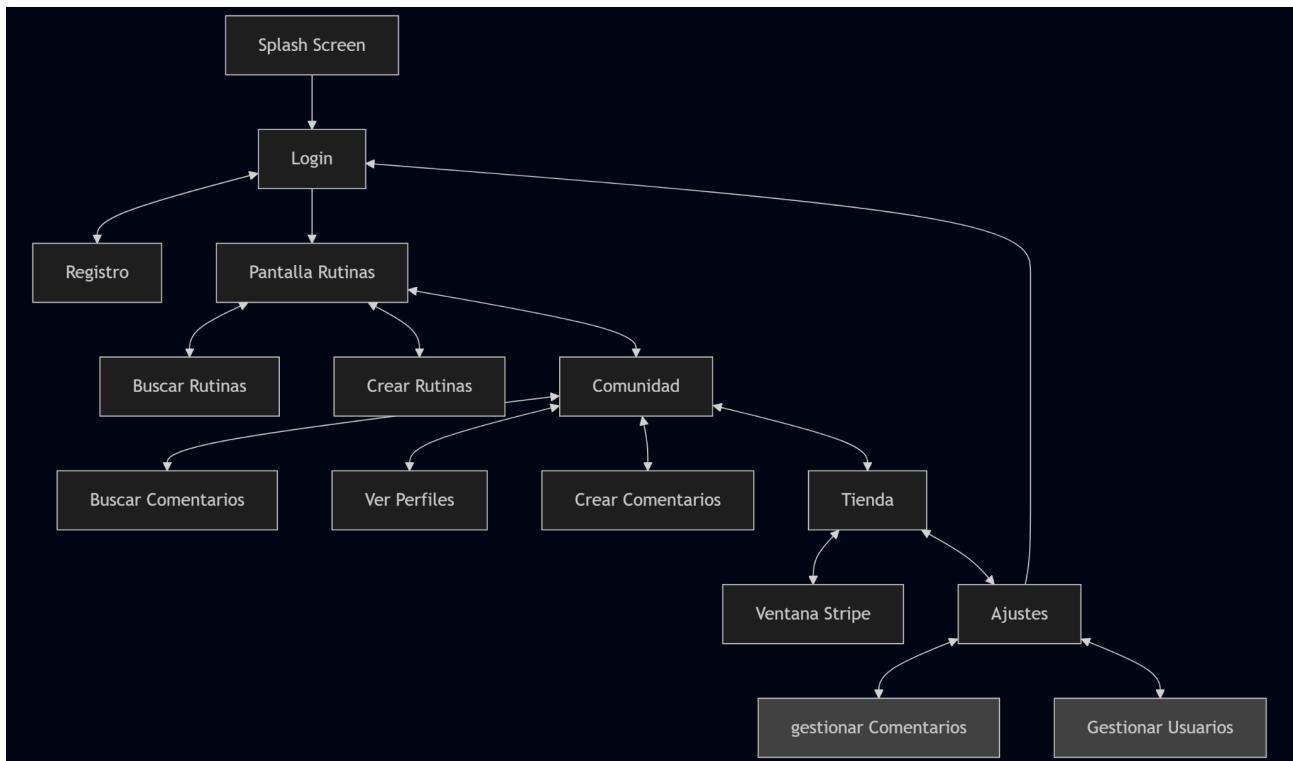


Como vemos solo el propio usuario puede actualizar sus estadísticas.

## 5.15 Cliente móvil

El cliente móvil de Rutify representa la interfaz principal de interacción entre el usuario y la plataforma. Está desarrollado en Kotlin utilizando Jetpack Compose, lo que permite una construcción declarativa y modular de las pantallas, facilitando la escalabilidad del proyecto.

### 5.15.1 Grafo de navegación



### 5.15.1 Arquitectura general

La aplicación sigue una arquitectura basada en MVVM (Model-View-ViewModel), permitiendo separar la lógica de presentación de la lógica de negocio. Esto mejora la mantenibilidad y testabilidad del código.

### **5.15.2 Flujo Registro**

**Registro de usuario:**

**Pantalla:** Registro

**ViewModel:** RegistroViewModel

**Ruta de navegación:** Rutas.Registro → Rutas.Login

**Flujo funcional:**

- Inicio del formulario de registro:
- Se muestra un formulario con los siguientes campos:
  - Nombre completo
  - Correo electrónico
  - Contraseña y confirmación
  - Fecha de nacimiento
  - Sexo (selector)
  - Aceptación de términos (checkbox)

**Validaciones locales en tiempo real:**

El **nombre** no puede estar vacío.

El **correo** debe tener formato válido.

La **fecha** de nacimiento debe ser válida y el usuario debe tener al menos 16 años.

La **contraseña** y su **confirmación** deben coincidir.

Debe aceptarse el **checkbox** de **términos y condiciones**.

**Botón Registrarse:**

Al pulsar, se lanza **viewModel.registrarUsuario()**.

Si todos los **campos son válidos**:

Se **crea el usuario**.

Se guarda la información del usuario (DTO) en la base de datos mediante Retrofit.

En caso de éxito:

Se elimina la pantalla de registro del backstack.

Se navega automáticamente a la pantalla de inicio de sesión (Rutas.Login).

Otras opciones:

Enlace de "¿Ya tienes cuenta? Inicia sesión": Redirige a Rutas.Login.

Botón "Salir": Finaliza completamente la app con `activity.finishAffinity()`.

### 5.15.3 Flujo Login

**Inicio de sesión de usuario:**

**Pantalla:** Login

**ViewModel:** LoginViewModel

**Ruta de navegación:** Rutas.Login → Rutas.Rutina

**Flujo funcional:**

**Inicio de la pantalla de inicio de sesión:**

Se muestra un **formulario** con los **siguientes campos**:

- Correo electrónico
- Contraseña
- Enlace para recuperar contraseña
- Enlace para registrarse si no se tiene cuenta
- Botón "Salir" para cerrar la app

**Validaciones locales en tiempo real:**

- El **correo** debe tener un **formato válido**
- La **contraseña** no puede estar **vacía** y debe tener al menos **6 caracteres**
- Botón "Iniciar sesión":
- Al pulsar, se ejecuta **viewModel.iniciarSesion(correo, contraseña)**
- Si las **credenciales** son **válidas**:
- **Firebase Auth** intenta **autenticar** al **usuario**

**En caso de éxito:**

- Se muestra un **toast de confirmación**
- Se **navega a Rutas.Rutina** y se **elimina la pantalla de login del backstack**

**En caso de fallo:**

- Se muestra un **toast indicando el error** (usuario no encontrado, credenciales incorrectas o error general)
- Se muestra una ventana modal si los campos están vacíos
- Recuperación de contraseña:
- Al pulsar en "¿Has olvidado tu contraseña?":
- Se abre una ventana modal solicitando el correo
- Si el correo es válido, se envía un correo de recuperación mediante Firebase

**Navegación adicional:**

- Enlace "¿No tienes cuenta? Regístrate" → navega a Rutas.Registro
- Botón "Salir" → finaliza completamente la app con `activity.finishAffinity()`

#### 5.15.4 Flujo Rutinas

Visualización y paginación de rutinas:

Pantalla: Rutinas

ViewModel: BuscarRutinasViewModel

Ruta de navegación: Rutas.Rutina

Flujo funcional:

Al iniciarse la pantalla:

- Se ejecuta **obtenerRutinas()** dentro de **LaunchedEffect(Unit)**
- Se solicita la primera página de rutinas al backend usando Retrofit
- Las **rutinas** se almacenan en **listaRutinas** y se muestran en **PantallaBusquedaRutina**
- Se renderiza **TopBarBase** con **título "Rutinas"** y **botón de búsqueda (Rutas.buscarRutinas)**
- Se muestra **NavigationBarAbajoPrincipal** con la **pestaña activa Rutas.Rutina**
- **Botón flotante** con icono **Add**, que **navega a Rutas.CrearRutinas**

Paginación

- Cada vez que se llama a **obtenerRutinas()**:
- Se **incrementa \_pagina** en 1
- Si la **respuesta** es **exitosa**:
  - Se **agregan** los **resultados** a **listaRutinas**
  - Si **hay resultados**: se muestra **toast "Cargando más"**
  - Si **no hay más resultados**: se muestra **toast "No hay más resultados"**
  - Si ya **no hay más rutinas** y el **toast ya** fue **mostrado**: **no se hace nada**
  - Si **hay error** de **conexión**:
    - **Se muestra R.string.errorConexion**
    - Se marca **sinInternet = true**

Sin conexión

- Se activa **onReintentar**, que:
  - Ejecuta **reiniciarPaginacion()**
  - Llama de nuevo a **obtenerRutinas()**

Reinicio de paginación

Al llamar a **reiniciarPaginacion()**:

- Se **limpia listaRutinas**
- Se **resetea \_pagina** a 0
- Se marca **\_finalDeRutinas = false**

Navegación adicional

- **Botón de búsqueda** (icono Search): **Navega a Rutas.buscarRutinas**
- **Botón flotante**: **Navega a Rutas.CrearRutinas**

### 5.15.5 Flujo Detalles Rutina

Visualización, favoritos y gestión de una rutina específica:

- **Pantalla:** PantallaDetallesRutinas
- **ViewModel:** DetallesRutinasViewModel
- **Ruta de navegación:** Rutas.DetallesRutina (implícita por parámetro idRutina)

#### Flujo funcional

##### Inicio de la pantalla

Al iniciar la pantalla con un idRutina:

- En **LaunchedEffect(Unit)** se ejecutan simultáneamente:
- **viewModel.obtenerRutina(idRutina):** obtiene la rutina desde API REST con **Retrofit**
- **viewModel.inicializarBD(context):** inicializa base de datos local **Room**
- **viewModel.comprobarFavorito(context, idRutina):** comprueba si la rutina está **marcada como favorita en BD local**
- El **estado general (estado)** controla si la **pantalla** está **cargando o lista**
- **sinInternet** indica **fallo de conexión**
- **favorito** indica si la rutina está **guardada localmente** como favorita
- **esSuyaOEsAdmin** habilita **acciones especiales (borrar rutina)** si el usuario es el creador o admin
- Se observa la rutina cargada en rutina para mostrar detalles

##### Renderizado UI

- **PantallaBase** muestra la **UI principal** con:
- **TopBarBase** con:
  - **Botón atrás** (flecha) que **navega atrás** en el stack
  - **Icono favorito** (estrella llena o borde) para **marcar/desmarcar rutina favorita**
  - **Icono eliminar** (papelera) **visible sólo si esSuyaOEsAdmin es true**
- **Cuerpo:**
  - **RutinasCard** que **despliega**:
  - **Título de la rutina**
  - **Imagen de la rutina** (con icono personalizado y bordes redondeados)
  - **Acordeón** con descripción, ejercicios (lista nombres), y equipo necesario
  - **Botón principal "Empezar" habilitado según estado**, que **inicia la rutina y navega a Rutas.HacerEjercicio**
- Si **ventanaEliminarRutina** es **true**, se **muestra AlertDialogConfirmar** para **confirmar borrado**

#### Interacciones y acciones

- Marcar/Desmarcar favorito
- Al tocar icono estrella se alterna estado favorito
- Si pasa a favorito: se guarda rutina en BD local Room y se muestra toast "Rutina guardada"
- Si pasa a no favorito: se elimina de BD local y se muestra toast "Rutina no guardada"

### **Eliminar rutina**

- Al tocar ícono papelera, se muestra diálogo confirmación
- **Confirmar llama a API para eliminar la rutina remotamente**
- Si éxito, se muestra toast "Rutina eliminada", se navega a Rutas.Rutina limpiando el backstack
- Si falla, se muestra toast con error

### **Reintentos**

- En caso de fallo de conexión, se activa sinInternet
- Se ofrece función de reintentos que vuelve a llamar a obtener rutina, inicializar BD y comprobar favorito

### **Empezar rutina**

- Al pulsar "Empezar", se cargan ejercicios de la rutina en el estado global ente
- Se navega a pantalla de ejecución de rutina Rutas.HacerEjercicio

### **Manejo de conexión**

- Si hay error de conexión durante llamadas API:
  - Se muestra mensaje de error (R.string.errorConexion)
  - Se marca sinInternet = true para activar el mecanismo de reintentos

### 5.15.6 Búsqueda online de rutinas

**Pantalla:** PantallaBusquedaRutinasOnline

**ViewModel:** BusquedaRutinasOnlineViewModel

**Ruta de navegación:** Rutas.buscarRutinas

#### Flujo funcional:

##### Inicio de la pantalla

- Se crea instancia del ViewModel **BusquedaRutinasOnlineViewModel**.
- Se obtienen los valores observados:
  - **textoBusqueda** (cadena de texto para buscar rutinas)
  - **estado** (booleano para estado de carga/completado)
  - **listaRutinas** (lista con las rutinas que se muestran)
  - **sinInternet** (booleano para estado de conexión)

##### Se muestra la PantallaBase con:

- **topBar personalizado** con título "Buscar rutinas" y botón de volver.
- Indicadores de carga y sin conexión según estado.
- Contenido:
  - **Campo de texto** para ingresar el texto de búsqueda.
  - **Botón con ícono de búsqueda** para disparar la consulta.
  - Componente **PantallaBusquedaRutina** que muestra los resultados.

##### Búsqueda y actualización de datos:

- Cuando el usuario escribe en el campo, se actualiza **textoBusqueda** en el **ViewModel** con **onNombreBusqueda**.
- Al pulsar el ícono de búsqueda, se llama a **buscarRutinas** con el texto actual.
- **buscarRutinas:**
  - Cambia estado de carga **\_estado** a false.
  - Intenta obtener las rutinas que coinciden mediante Retrofit.
  - Si la llamada es exitosa, actualiza la lista **\_listaRutinas**.
    - eCambia stado de carga a true.
    - Si hay error de conexión, activa **\_sinInternet** y maneja el error.

##### Sin conexión

Si **\_sinInternet** es true, PantallaBase mostrará la pantalla **SinConexionPantalla**.

##### Navegación adicional

Botón de flecha atrás en el topBar para volver con **navControlador.popBackStack()**.

### **5.15.7 Flujo actividad del usuario**

Visualización de rutinas creadas, comentarios, votos y rutinas favoritas

**Pantalla:** Actividad

**ViewModel:** RutinasFavoritasViewModel

**Ruta de navegación:** Rutas.rutinasFavoritas

#### **Flujo funcional**

##### **Inicio de la pantalla**

Al abrir la pantalla Actividad, se muestra una top bar con opción para volver.

- Se cargan distintos **datos según la pestaña activa**:
- **0: Rutinas creadas (desde backend)**
- **1: Comentarios del autor**
- **2: Votos realizados**
- **3: Rutinas favoritas (desde base de datos local)**

##### **Paginación**

Actualmente no hay paginación real.

##### **Sin conexión**

- Usa **sinInternet** del **ViewModel** para mostrar una **pantalla sin conexión**.

##### **Pull To Refresh**

- **Implementado con PullToRefreshBox en PantallaBusquedaRutinaLocal.**
- Al hacer **swipe down**, se llama a **viewModel.obtenerRutinasFavoritas(context)**.

##### **Navegación adicional**

- Al **pulsar una rutina**, se navega a "rutinas/{id}".
- En **votos**: cada **tarjeta** tiene un **RatingBar editable** y un **botón para eliminar**.
- En **comentarios**: puedes **eliminar uno con confirmación** vía **AlertDialogConfirmar**.

### **5.15.8 Flujo Crear rutina personalizada**

Flujo de creación de una rutina con ejercicios personalizados, descripción, ícono, calorías y puntos.

**Pantalla:** CrearRutinas

**ViewModel:** CrearRutinasViewModel

**Ruta de navegación:** Rutas.CrearRutina

#### **Flujo funcional**

##### **Inicio de la pantalla**

- Se llama a **viewModel.obtenerEjerciciosDesdeApi()** al iniciar la pantalla.
- Se observa **sinInternet** para mostrar la pantalla de error si falla la carga.
- PantallaBase con TopBarBase permite retroceder usando el ícono de back.

##### **Edición de información**

- **Icono de rutina:** Toca el ícono actual → **viewModel.abrirVentanaImagenes()** → ventana emergente con selección.
- **Nombre:** Campo de texto con límite (limiteNombre) y contador.
- **Descripción:** Campo de texto expandido con scroll y contador (limiteDescripcion).

##### **Ejercicios**

- Se muestra un LazyColumn con tarjetas (ItemCantidad) por ejercicio.
- Cada ejercicio permite modificar la cantidad y eliminarse.
- Botón Añadir ejercicio abre un BottomSheet o modal (mostrarEjercicios).
- Puedes ver detalles de cada ejercicio (→ mostrarDetalleEjercicio).
- Puedes añadir ejercicios desde los detalles (viewModel.anadirEjercicioALaLista()).
- Confirmación para eliminar ejercicio (mostrarVentanaEliminarEjercicio).

##### **Equipo**

LazyRow que muestra tarjetas con nombres del equipo requerido (listaEquipo).

##### **Métricas**

- Tarjetas con:
  - Calorías estimadas.
  - Puntos ganados.

##### **Creación**

- Botón Crear rutina:
  - Llama a **viewModel.crearRutina { estado -> ... }**.
  - Navega a Rutas.Rutina si se creó correctamente.

### 5.15.9 Flujo realizar rutina

**Pantalla:** HacerEjercicioRutina

**ViewModel:** EjercicioRutinasViewModel

**Ruta de navegación:** Rutas.HacerEjercicio

**Flujo funcional:**

Inicio de la pantalla

Se lanzan varias operaciones al entrar:

- **cargarEjercicio():** carga el primer ejercicio desde una lista global ente.listaEjercicio.value!!.
- **obtenervoto():** obtiene el voto actual del usuario para esa rutina (si ya ha votado).
- **obtenerEstadisticas():** obtiene las estadísticas actuales del usuario.
- **iniciarTemporizador():** comienza un contador de segundos en segundo plano.
- Si el estado finalizado cambia, se detiene el temporizador.

**Ejecución de ejercicios**

- Se muestra la vista pantallaHacerejercicio si
  - !finalizado && !cancelado.
- Al pulsar "Siguiente":
  - Se llama a siguienteEjercicio().
  - Si hay más ejercicios en la lista:
    - Se incrementa el contador y se carga el nuevo ejercicio.
  - Si es el último ejercicio:
    - Se establece finalizado = true.
    - Llama a calcularProgreso() para preparar las estadísticas acumuladas de la rutina.

**Cálculo de progreso (calcularProgreso())**

Se suman:

- ejerciciosRealizados += 1 por cada ejercicio.
- kCaloriasQuemadas se calcula como (cantidad \* calorías por repetición) / 1000.
- Se suman los puntos ganados al grupo muscular correspondiente (brazo, pecho, abdominales, etc).

**Finalización**

Si finalizado es true:

- Se muestra la pantalla pantallaFinal con el resumen.
- El botón de confirmar llama a:
  - guardarEstadisticaDiaria()
  - guardarProgreso() → si éxito, se hace navControlador.popBackStack().

**Votación**

- Si ventanaPuntuarRutina es true, se muestra el PuntuarRutinaDialog:
- onConfirm: guarda el voto con guardarVoto().
- onDismiss: oculta la ventana con VentanaPuntuarRutina(false).

**Sin conexión**

En caso de excepción en llamadas de red (obtenervoto, obtenerEstadisticas, etc):

Se marca sinInternet = true.

Se muestra un toast R.string.errorConexion.

**Navegación adicional**

- Si el **usuario cancela la rutina** (cancelado == true):
  - Se hace `popBackStack()`.
  - El **botón de puntuar solo aparece tras finalizar.**

#### Lógica del ViewModel

##### Datos cargados

- `ente.listaEjercicio.value!!`: contiene la lista de **ejercicios** de la rutina actual.
- `ente.rutina.value!!`: ID de la **rutina actual**.

##### Guardado de progreso

###### En `guardarProgreso()`:

- Se **acumulan los valores** de `estadisticasDtoCalculadas` a `estadisticasDto`.
- Se **actualiza o crea** en la API (`actualizarEstadisticas` o `crearEstadisticas`).

##### Temporizador

- Se **gestiona** con una `Job` que **actualiza \_tiempo** cada segundo.
- Se **cancela si finaliza** la rutina.

#### **5.15.10 Pantalla del perfil del usuario**

##### Mi Zona – Seguimiento personal, avatar y reto diario

**Pantalla:** MiZona

**ViewModel:** MiZonaViewModel

**Ruta de navegación:** Rutas.MiZona

**Flujo funcional**

**Inicio de la pantalla:**

Se inicializan múltiples LiveData observables con valores por defecto.

Se ejecuta un **LaunchedEffect** que llama a:

- obtenerCountRutinasFavoritas
- iniciarContadorTiempoRestante
- obtenerUltimos5Pesos
- obtenerUsuario
- obtenerComesticosComprados
- obtenerEstadisticasDiaria
- obtenerObjetivosLocal

**Avatar del usuario**

- Se muestra con **cuadroAvatar**, junto a opciones:
  - Cambiar icono de avatar
  - Cambiar ropa (cosméticos)

**Estadísticas del día**

- Se muestran en tarjetas clicables que redirigen a **Rutas.Estadisticas**.
  - Datos presentados:
    - Calorías quemadas vs meta (metaKcal)
    - Ejercicios realizados vs meta (metaRutinas)
    - Horas activo vs meta (metasMinActivos)

**Peso del usuario**

- Se muestra la última medición y un gráfico con los últimos **5 pesos** (ultimosPesos).
  - Redirige a estadísticas diarias.

**Información adicional**

- Tarjeta con accesos a:
- Rutinas favoritas (**countRutinasFavoritas**)
- Rutinas creadas
- Comentarios publicados
- Votos realizados

**Reto diario**

- Botón que abre un reto si no ha sido completado hoy (usuario.fechaUltimoReto).

- Si ya fue completado, muestra un toast.
- Al **iniciar**, se **muestra** una **ventana modal** con:
  - **Descripción del ejercicio**
  - **Temporizador**
  - **Botón para completar**

#### Ventana cambiar icono de avatar

- **Modal** que **muestra imágenes** del **avatar disponibles** (imagenes).
- Al **seleccionar**, se **cambia el avatar** y se **cierra la ventana**.

#### Ventana cambiar ropa

- **Modal** que **muestra cosméticos agrupados por tipo** (cosmeticosPorTipo) usando **MostrarCosmeticosLazyRows**.
- Al **pulsar equipar**, se **aplica el cosmético** y se **cierra la ventana**.

### **5.15.11 Pantalla estadísticas diarias del usuario**

Visualización, edición y navegación de estadísticas personales por día

**Pantalla:** Estadisticas

**ViewModel:** EstadisticasViewModel

**Ruta de navegación:** Ruta.Estadisticas

**Flujo funcional:**

#### Inicio de la pantalla

- Al **iniciarse**, se **obtiene la fecha seleccionada** (LocalDate.now() por defecto).
- Se **llama a cambiarFecha(fecha)** que **invoca obtenerEstadisticas()** para **recuperar las estadísticas del día seleccionado** desde la API.
- Se **observa el estado de carga** (estado), de **conexión** (sinInternet), y se **visualiza** un **PantallaBase** con **topBar**, **botón flotante**, y **tarjeta con contenido**.

#### Contenido principal

- **Flechas a izquierda/derecha** para **cambiar de día** (sumar/restar un día a la fecha).
- Se **muestran 4 tarjetas** si existen **estadísticas**:
- **Ejercicios** realizados.
- **Calorías** estimadas quemadas.
- **Horas** activas.
- **Peso** corporal.
- Si no **existen estadísticas** (\_id == null), se **muestra el mensaje "No existen mediciones"**.

#### Registro de peso corporal

- El **FAB** (botón flotante con ícono de peso) abre un **AlertDialog** para **registrar el peso del día seleccionado**.
- **Campo de texto validado con expresión regular** para **evitar valores no numéricos**.

#### **Botón "Guardar":**

- **Llama a guardarPeso()**, que hace **PATCH** a la **API** con un **DTO (EstadisticasDiariasPatchDto)** y **actualiza la estadística del día**.
- Luego **vuelve a llamar a obtenerEstadisticas()** para **refrescar**.
- **Botón "Cancelar"** cierra el diálogo sin guardar.

#### **Paginación de días**

- **Botones permiten retroceder o avanzar un día** con **cambiarFecha(fecha.minusDays(1))** o **fecha.plusDays(1)**.

#### **Sin conexión**

- Si **ocurre un error de red** (API no disponible, sin internet), se activa **\_sinInternet**.
- Se **muestra la pantalla de error de PantallaBase** y el **botón de "Reintentar"** **vuelve a llamar a reiniciarInternet() y cambiarFecha()**.

#### **Reinicio de estado**

- Cada vez que se cambia de día o se guarda una estadística:
  - **estado** se pone en **false** → se muestra loading.
  - Luego de la respuesta exitosa, se actualiza **estado = true**.

#### **Navegación adicional**

- Icónico de volver (topBar) → **navControlador.popBackStack()**.

### 5.15.12 Pantalla perfil

Pantalla de visualización del perfil de un usuario desde la comunidad

**Pantalla:** perfil (Composable)

**ViewModel:** PerfilViewModel

**Ruta de navegación:** Rutas.Perfil

**Flujo funcional:**

#### Inicio de la pantalla

- Se lanza un **LaunchedEffect** al **entrar en la pantalla**:
  - viewModel.**obtenerIdFirebase(idFirebaseParam)**
  - viewModel.**obtenerUsuario()**
  - viewModel.**obtenerUltimos5Pesos()**

#### Información mostrada

- Si el **perfil es privado** (perfilprivado):
  - Se **muestra** una **tarjeta** informando que el **perfil es privado** (R.string.perfilPrivado)
  - No se muestra más contenido
- Si el **perfil no existe** (noExiste):
  - Se muestra una **tarjeta** informando que el **perfil no se encuentra** (R.string.noEncontrado)
  - No se muestra más contenido
- Si el **perfil existe y es público**:
- Se **muestra**:
  - **Avatar** (cuadroAvatar(usuario))
  - **Últimos 5 pesos** en gráfica (PesoGraph)
  - **Cantidad de rutinas** creadas
  - **Cantidad de comentarios**

#### Navegación adicional

- Al **tocar la tarjeta de rutinas** → **navega** a "rutinasFavoritas/{idFirebase}"
- Al **tocar la tarjeta de comentarios** → también **navega** a "rutinasFavoritas/{idFirebase}"

#### Estados manejados

- **estado**: controla la **visibilidad** del **spinner** de carga
- **sinInternet**: controla si mostrar la **pantalla** de **error sin conexión**
- **noExiste**: si el **usuario no está registrado** (error 404)
- **perfilprivado**: si el **usuario ha marcado su perfil como privado** (error 401)

#### Sin conexión

- Si **ocurre una excepción**, se **muestra** un **Toast** con R.string.errorConexion
- Se **actualiza \_sinInternet** para que **PantallaBase** muestre **pantalla correspondiente**

#### Reintento

- Si el **usuario** pulsa **reintentar** en **pantalla** de sin conexión:
  - Se vuelve a **ejecutar** viewModel.**obtenerUsuario()**

### 5.15.13 Pantalla foro

Comentarios generales (padres)

**Pantalla:** Foro

**ViewModel:** ForoViewModel

**Ruta de navegación:** Rutas.Comunidad

**Flujo funcional:**

**Inicio de la pantalla**

- Al cargar la pantalla se ejecutan:
  - obtenerUsuario()
  - comprobarAdmin()
  - obtenerComentarios()

**Crear comentario**

- Al pulsar el botón flotante  se abre el dialogoCrearComentario.
- Se permite:
  - Escribir un texto
  - Elegir estado de ánimo (estadoAnimoSeleccionado)
  - Subir imagen desde galería (launcher) o tomar foto (takePictureLauncher)
- La imagen se comprime a 100KB antes de enviarse al backend (comprimirImagenA100KB).
- Se construye un objeto ComentarioDto con:
  - ID Firebase del usuario actual
  - Nombre y avatar del usuario
  - Fecha actual
  - Texto del comentario
  - Estado de ánimo
- Si la imagen existe, se adjunta como MultipartBody.Part.
- Se realiza la petición al endpoint apiComentarios.createComentario.

**Eliminar comentario**

- Si se pulsa eliminar en un comentario:
  - Se guarda en comentarioAEliminar
  - Se muestra AlertDialogConfirmar
  - Si se confirma, se borra el comentario (borrarComentario())

**Listado de comentarios**

Se observan los comentarios con LiveData (comentarios)

Se muestran con PantallaComentarios

Se distingue si es administrador o autor del comentario para mostrar opciones de eliminación

**Manejo de imagen**

Para galería: ActivityResultContracts.GetContent()

Para cámara:

Se crea un Uri temporal con crearUriParaFoto

Se lanza TakePictureLauncher

**Sin conexión**

Si falla la carga de comentarios: sinInternet = true

Se muestra la UI de reintentar:

Vuelve a llamar a obtenerUsuario, comprobarAdmin y obtenerComentarios

### **5.15.14 Pantalla detallesComentario**

**Pantalla:** DetallesComentario

**ViewModel:** DetallesComentariosViewModel

**Ruta de navegación:** "Pantalla.DetallesComentario/<IdComentario>"

**Flujo funcional:**

**Inicio de la pantalla**

**LaunchedEffect(Unit)** inicializa:

- viewModel.**obtenerUsuario()** → **obtiene el usuario** actual.
- viewModel.**comprobarAdmin()** → **determina si es admin** o autor del comentario.
- viewModel.**obtenerComentarioPadre(urlComentario)** → **guarda el ID del comentario padre.**
- viewModel.**obtenerComentarios()** → **obtiene el comentario padre + respuestas.**

**Mostrar comentario padre**

- Se usa PantallaComentarios para mostrar solo el comentario padre en un bloque con height(150.dp) y maxImagen = 50.dp.

**Escribir una respuesta**

Campo de texto expandible de hasta 150.dp con scroll.

- **Botón "Publicar"** llama a **crearComentario()**, que:
  - **Usa los datos del usuario + texto actual.**
  - **Llama a apiComentarios.responderComentario.**
  - **Limpia el texto y actualiza la lista (obtenerComentarios()).**

**Mostrar respuestas**

- Segundo PantallaComentarios muestra la lista completa de comentarios (hijos).

**Eliminar comentario**

Si el comentario es propio o el usuario es admin:

- **Aparece el ícono de eliminar.**
- **Llama a guardarComentarioAEliminar() y muestra la ventana AlertDialogConfirmar.**
- En **borrarComentario**, si el **comentario eliminado** es el **padre**, se **hace popBackStack()** (volver).

**Reportar usuario**

- El **botón de reportar (ícono superior)** abre ventana de confirmación.
- Llama a **reportarUsuario()** que:
  - Llama a **apiReportes.reportarUsuario(idFirebaseDelPadre)**.
  - Muestra **Toast** según el código:
    - **201: Reportado.**
    - **409: Ya reportado.**
    - **404: Usuario no encontrado.**
    - Otro: Error genérico.

## **Sin conexión**

- Si ocurre error de red (Exception), se activa `_sinInternet = true` y se muestra la pantalla correspondiente con opción de reintentar (onReintentar).

## **Navegación adicional**

- **Icono "volver"** llama a `navControlador.popBackStack()`.
- Al eliminar el comentario padre, también se vuelve hacia atrás automáticamente.

### **5.15.15 Pantalla Buscador**

Pantalla con búsqueda por nombre de usuario o autor de comentario, con posibilidad de eliminar comentarios si el usuario es dueño o admin.

**Pantalla:** Buscador

**ViewModel:** BuscadorViewModel

**Ruta de navegación:** Ruta.Buscador

#### **Flujo funcional:**

##### **Inicio de la pantalla**

- Muestra una **TopBarBase** con **botón para volver**.
- Muestra una **TarjetaNormal** con:
  - **Campo de texto** con **ícono de búsqueda**.
  - **Dos pestañas (TabRow)**: "Buscar cuentas" y "Buscar comentarios por autor".
- Al **cambiar de pestaña**, se **lanza buscar()** automáticamente

##### **Búsqueda**

- La **búsqueda** se ejecuta manualmente al pulsar el ícono de **búsqueda**.
- Si la **pestaña activa** es **0**, se **llama a buscarCuentasPorNombre()**.
- Si la **pestaña activa** es **1**, se **llama a buscarComentariosPorAutor()**.

##### **Visualización de resultados**

- Si la **pestaña** es **0**, se **muestra la lista de usuarios (PantallaBusquedaUsuarios)**.
- Si es **1**, se **muestra PantallaComentarios**, que **permite**:
  - **Ver los comentarios**.
  - **Eliminar un comentario** si es **suyo** o es **admin**.

##### **Eliminación de comentario**

- Al **hacer clic en eliminar**:
- Se **guarda el comentario** en `_comentarioAEliminar`.
- Se **muestra AlertDialogConfirmar**.
- Si se **acepta**:
  - Se **llama a borrarComentario()**.
  - Se **elimina de la lista local** y se **muestra toast**.

### **5.15.15 Pantalla Ajustes**

Configuración de preferencias del usuario y funciones administrativas

**Pantalla:** Ajustes

**ViewModel:** SettingsViewModel

**Ruta de navegación:** Rutas.Ajustes

#### **Flujo funcional**

##### **Inicio de la pantalla**

Se lanza LaunchedEffect(Unit):

- viewModel.obtenerUsuario()
- viewModel.comprobarAdmin()

Se observan los siguientes estados desde el ViewModel:

- settings
- esAdmin
- fontSizes, fontLabels, themeOptions
- popupVentanaEliminarCUenta
- estado (cargando)
- sinInternet

#### **Contenido visual**

- **PantallaBase** con:
  - **BottomBar:** NavigationBarAbajoPrincipal
  - **Estado de carga** (estado)
  - **Estado de conexión** (sinInternet)
- **Dentro de TarjetaNormal:**
  - **Tamaño de fuente:**
    - **Lista de radios con textos: Pequeño / Normal / Grande**
    - **Al seleccionar:** viewModel.updateSettings con el **nuevo fontSizeScale**
- **Tema:**
  - **Radios: Seguir sistema / Claro / Oscuro**
  - **Al seleccionar:** viewModel.updateSettings con el **nuevo themeOption**

#### **Sesión y Cuenta**

- **Botón “Cerrar sesión”:**
  - **FirebaseAuth.getInstance().signOut()**
  - **Navegación a Rutas.Login con popUpTo inclusivo**

#### **Botón “Eliminar cuenta”:**

- **Abre AlertDialogConfirmar si popupVentanaEliminarCUenta == true**
- **Al confirmar:**
  - **viewModel.eliminarCuenta { FirebaseAuth.getInstance().signOut(); navegar a Login }**
  - **Se elimina con API RetrofitClient.apiUsuarios.eliminarCuenta(...)**

### Zona Administrador

- Solo visible si esAdmin == true
- Botones visibles:
  - Administrar Comentarios → Rutas.administrarComentarios
  - Administrar Usuarios → Rutas.administrarUsuarios

### Lógica en ViewModel

- updateSettings(newSettings):
  - Guarda settings en Room y actualiza LiveData
- mostrarPopUpEliminarCuenta():
  - Alterna el estado booleano de la ventana de confirmación
- eliminarCuenta():
  - Llama a la API y gestiona navegación y errores
  - Muestra toast si hay error de conexión

## 6. Desafíos y problemas afrontados

Resumen de incidencias afrontadas y soluciones

Incidencia Técnica	Solución Aplicada
No sincronización inicial de ajustes	Carga con Room en init {} y LiveData observable
Possible pérdida de ajustes al cambiar valores	Persistencia inmediata con updateSettings()
Fallos de red al eliminar cuenta	Try/catch + mensaje + estado de sin conexión
Eliminación accidental de cuenta	Confirmación doble con AlertDialogConfirmar
Acceso no autorizado a funciones de admin	Verificación previa con comprobarAdmin()
Errores al seleccionar opciones (RadioButton)	Uso correcto de selectable y forEachIndexed
Pantalla cargando eternamente	Uso de estado y PantallaBase() para control de carga

## 7. Desarrollo

La aplicación **Rutify** ha sido **desarrollada** utilizando una **arquitectura moderna y escalable** basada en tecnologías robustas y ampliamente adoptadas tanto en el desarrollo de aplicaciones móviles como en backend. A continuación, se detallan los componentes principales del sistema:

### Arquitectura general

La arquitectura de Rutify sigue un **enfoque en capas y patrones bien definidos** para garantizar la mantenibilidad, escalabilidad y estabilidad de la aplicación.

### Backend

#### Framework: Spring Boot

Se encarga de la **lógica del negocio**, la exposición de **APIs RESTful** y la **conexión con la base de datos MongoDB**. Proporciona **inyección de dependencias**, manejo de excepciones, seguridad, validaciones y serialización de datos.

#### Base de datos: MongoDB

Utilizada como base de datos NoSQL para almacenar información estructurada como usuarios, rutinas, estadísticas, comentarios y más.

#### Correo electrónico: Simple Java Mail

Biblioteca empleada para **enviar correos electrónicos transaccionales**, como confirmaciones de registro o notificaciones importantes.

### Frontend

#### Lenguaje y Framework: Kotlin con Jetpack Compose

La interfaz de usuario está construida con Jetpack Compose, el moderno toolkit de UI declarativo de Android. Facilita una experiencia de desarrollo fluida, componentes reutilizables y una integración directa con el ciclo de vida del sistema Android.

### Seguridad

#### Proveedor de autenticación: Firebase Authentication

Utilizado para gestionar la autenticación de usuarios (registro, inicio de sesión, recuperación de contraseña) mediante email/contraseña y otros métodos. El token de Firebase se valida en cada petición protegida al backend para garantizar que el usuario está autenticado.

### Gestión de imágenes

#### Cloudinary

Servicio utilizado para el **almacenamiento y optimización de imágenes** (por ejemplo, avatares de usuarios, fotos en comentarios). Las imágenes se suben desde el frontend y se almacenan de forma externa para optimizar el rendimiento de la app.

## 8. Pruebas

La aplicación Rutify ha sido sometida a diferentes tipos de pruebas para garantizar su correcto funcionamiento, tanto a nivel técnico como a nivel de experiencia de usuario.

### Pruebas unitarias (Backend)

Se han implementado pruebas unitarias sobre el backend desarrollado con Spring Boot, enfocándose en los servicios y la lógica de negocio. Estas pruebas permiten verificar que los métodos funcionan correctamente de manera aislada, utilizando herramientas como **JUnit** y **Mockito** para simular dependencias y asegurar que los resultados son los esperados en distintos escenarios.



Imagenes: Pruebas unitarias.

> controller	0% (0/13)	0% (0/62)	0% (0/82)	100% (0/0)
> domain	80% (12/15)	83% (54/65)	85% (163/1...	0% (0/2)
> dto	90% (18/20)	90% (18/20)	94% (115/1...	100% (0/0)
> exception	62% (5/8)	35% (5/14)	25% (5/20)	0% (0/4)
> iService	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
> repository	100% (1/1)	100% (1/1)	100% (1/1)	100% (0/0)
> service	94% (18/19)	85% (86/101)	80% (504/...	65% (208/...
> utils	66% (2/3)	55% (11/20)	61% (92/15...	10% (1/10)

### Pruebas funcionales y de usuario (Frontend)

Tests	Duration	Pixel_7a_API_35
✓ Test Results	59 s	1/1
✓ RegistroTest	59 s	1/1
✓ usuario_se_registro	59 s	✓

Hice una prueba unitaria en el registro, que es la pantalla que menos testeada esta por parte de mis familiares

Para validar la experiencia de usuario y el correcto funcionamiento de la aplicación móvil, se realizaron pruebas con usuarios reales. En concreto, mis padres y mi novia, que utilizaron la aplicación en condiciones reales, proporcionando feedback sobre la usabilidad, navegación y diseño de la interfaz.

En general, las pruebas a mis padres y a mi novia, fueron óptimas.

## 9. Distribución

La distribución de la aplicación Rutify se realizará de la siguiente manera:

**Backend:** El backend desarrollado en **Spring Boot** será desplegado en la **plataforma Render**, que proporciona una infraestructura en la nube escalable y de fácil gestión, ideal para proyectos en fase de desarrollo y producción.

**Aplicación móvil (APK):** Inicialmente, el archivo **APK de la aplicación estará disponible** para descarga a través de:

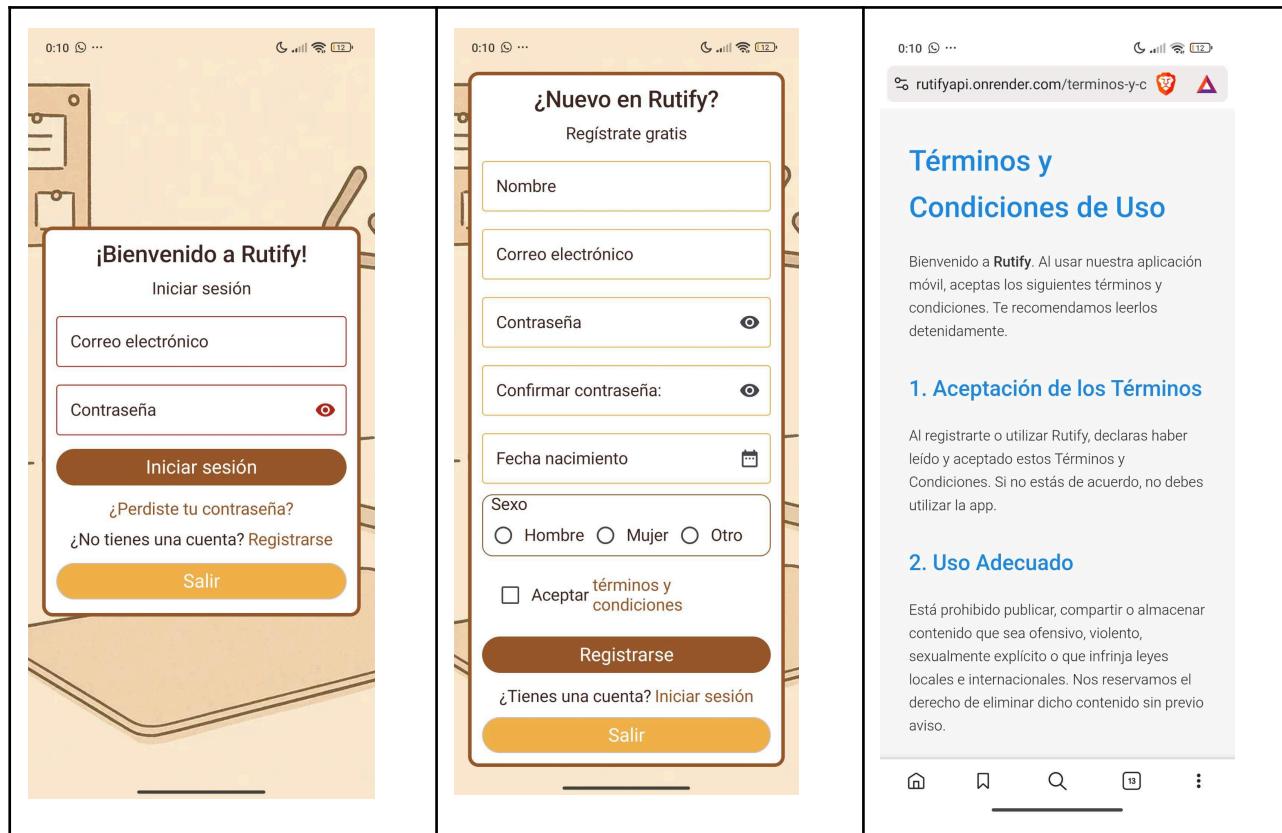
La **página web oficial del proyecto**.

Un **enlace compartido mediante Google Drive**, asegurando un acceso fácil y rápido para los primeros usuarios y testers.

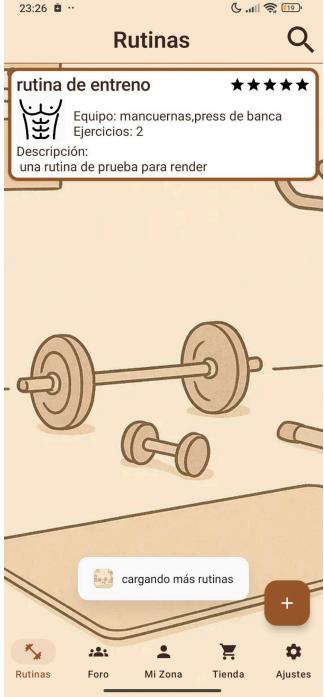
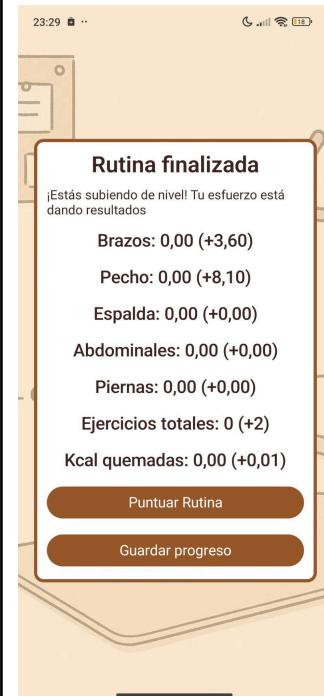
**Publicación en Google Play:** Está previsto que en una fase futura, una vez completado el proceso de validación y pruebas finales, la aplicación sea publicada en la Google Play Store para su distribución masiva.

Esta estrategia permite una **distribución progresiva que facilita** la recolección de **feedback temprano** antes de un lanzamiento completo en plataformas oficiales.

# 10. Manual



Como vemos tengo un inicio de sesión y un registro, bastante simple, con sus términos y condiciones para no tener una comunidad tóxica, si el usuario es menor a 16 años no se puede registrar.

 <p>Imagen: pantalla buscar rutinas</p>	 <p>Imagen: pantalla crear rutina</p>	 <p>Imagen: pantalla buscar rutinas</p>
 <p>Imagen: pantalla detalles rutinas</p>	 <p>Imagen: pantalla realizar rutina</p>	 <p>Imagen: pantalla rutina fina</p>

En el **módulo de rutinas**, el usuario podrá hacer rutinas, registrar estadísticas y tener un contador para saber cuánto tiempo le está llevando realizar la misma.



Imagen: pantalla mi zona



Imagen: pantalla reto diario



Imagen: pantalla estadísticas diarias



Imagen: pantalla equipar indumentaria

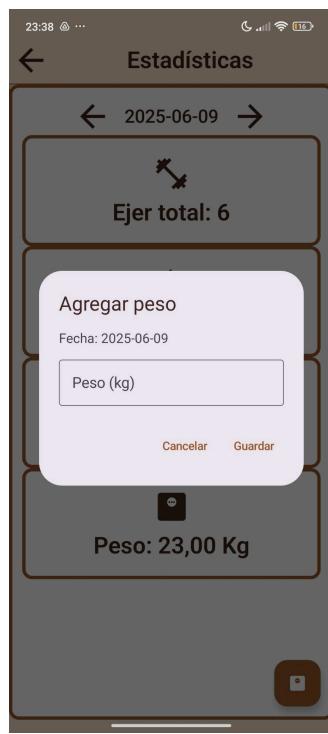


Imagen: pantalla estadistica anotar peso



Imagen: pantalla seleccionar foto avatar

En el **módulo mi zona**, el usuario podrá cambiarse de cara de avatar, cambiar la ropa del avatar, completar un reto diario que no puntúa en las estadísticas, anotar su peso y ver sus estadísticas.

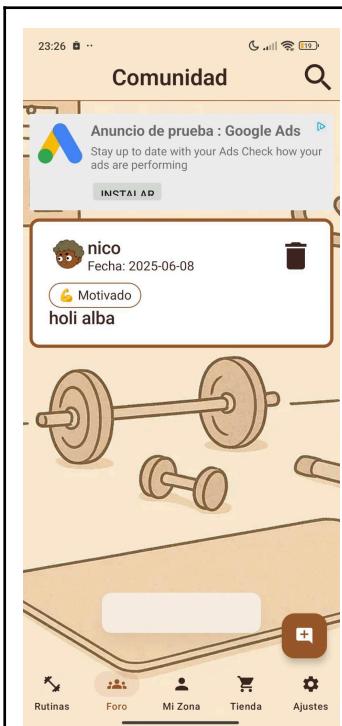


Imagen: pantalla buscar comentario



Imagen: pantalla ver comentario



Imagen: pantalla eliminar comentario

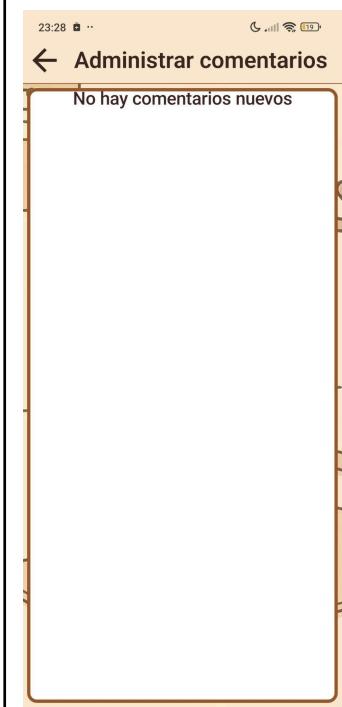


Imagen: pantalla administrar comentarios con fotos

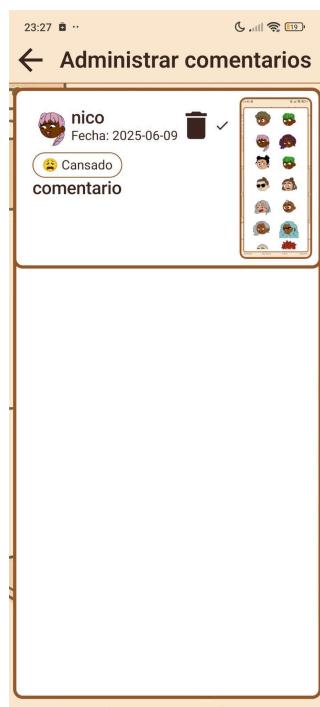


Imagen: pantalla administrar comentarios con fotos



Imagen: perfil de otro usuario

En el **módulo de comunidad**, los usuarios pueden ver los perfiles públicos de otras personas y comentarlos y responder, y eliminar sus propios comentarios, y los administradores tienen la tarea de aprobar comentarios con imágenes que no sean revisados

**Imagen: pantalla tienda**

**Imagen: pantalla comprar cosmético**

**Imagen: pantalla tienda**

**Imagen: pantalla stripe**

**Imagen: pantalla stripe tarjeta**

**Imagen: pantalla pago correcto**

En el **módulo de tienda**, podemos ver que el usuario con **dinero real**, puede adquirir monedas virtuales, que a su vez esas monedas virtuales las puede gastar en cosméticos para el avatar cambiando la apariencia del mismo.



Imagen: pantalla ajustes



Imagen: pantalla sin conexión

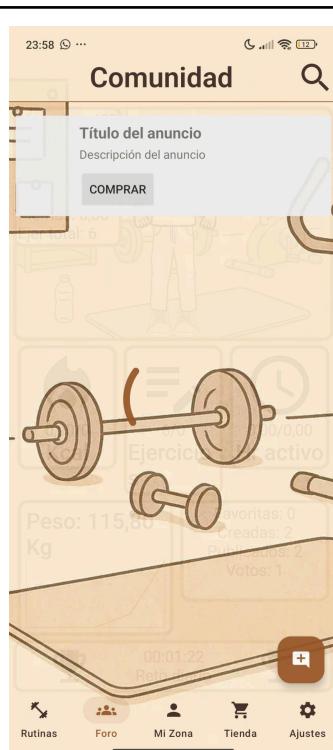


Imagen: pantalla cargando



Imagen: pantalla modo oscuro

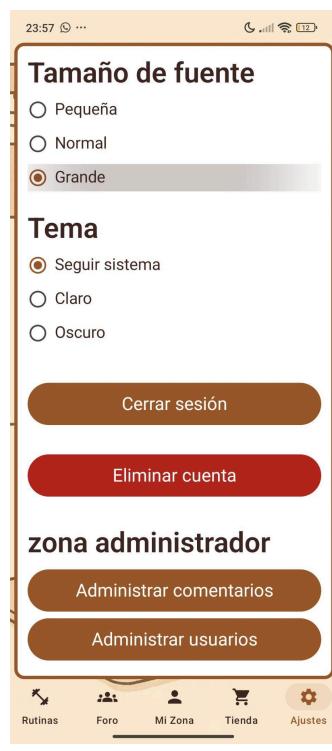


Imagen: pantalla texto grande

En el módulo ajustes, si eres administrador puedes administrar a usuarios con reportes y comentarios, también he añadido opciones de accesibilidad para usuarios mayores y he puesto el modo claro y oscuro que lo escogen a su antojo, también el usuario le indica cuando no tiene conexión a internet o esta la app cargando

## 11. Conclusiones

El desarrollo de la aplicación **Rutify** ha permitido crear una herramienta integral para el entrenamiento personal, diseñada especialmente para usuarios principiantes y aficionados al fitness, que buscan orientación clara y motivación para mejorar su salud y condición física.

Durante el desarrollo se aplicaron buenas prácticas en la arquitectura del software, con separación clara entre capas (controller, service, repository) y un patrón MVVM en el frontend, lo que facilita el mantenimiento y futuras ampliaciones del sistema y aumentar mi miedo al tener que manejar más de 10 archivos.

A pesar de las funcionalidades implementadas, se identifican áreas de mejora para futuras versiones, como la incorporación de un sistema de notificaciones para motivar al usuario, la integración de más métricas de salud, y la expansión hacia otras plataformas móviles o de sobremesa, con lo que tengo ahora estoy mas que satisfecho, haber logrado esto.

En conclusión, Rutify cumple con mis objetivos propuestos, proporcionando una aplicación funcional, intuitiva y motivadora que puede ayudar a muchas personas a iniciar y mantener hábitos de entrenamiento saludables, con un enfoque moderno y accesible

También gracias al equipo de docentes del Rafael Alberti por guiarme en esta aventura, a mis padres por aguantarme darle la lata para que prueben la app, y a mi novia Alba por aguantarme y darme ánimos e ideas de mejoras.

**Lo conseguí troy.**

## 12. Índice de tablas e imágenes

Imagen: Pantalla de inicio sesión.....	9
Imagen: Pantalla de registro.....	10
Imagen: Pantalla de búsqueda rutinas.....	11
Imagen: Pantalla detalles rutinas.....	12
Imagen: Pantalla de crear rutina.....	13
Imagen: Pantalla de realizar ejercicio.....	14
Imagen: Pantalla de mi zona.....	15
Imagen: Pantalla de reto diario.....	16
Imagen: Pantalla de elegir ejercicios.....	17
Imagen: Pantalla de configuración.....	18
Imagen: Pantalla de planes de pagos.....	19
Imagen: Entidad Relación.....	19
Tabla: Usuario.....	20
Tabla: UsuarioRegistroDTO.....	20
Tabla: UsuarioregistradoDto.....	21
Tabla: UsuarioCredencialesDto.....	21
Tabla: UsuarioLoginDto.....	21
Tabla: EliminarUsuarioDTO.....	21
Tabla: UsuarioBusquedaDto.....	21
Tabla: UsuarioinformacionDto.....	22
Tabla: Indumentaria.....	22
Tabla: EstadisticasDto.....	23
Tabla: ActualizarUsuarioDTO.....	23
Tabla: endpoints Usuarios.....	24
Tabla: Json E/S usuario se registra.....	25
Diagrama flujo: usuario se registra.....	25
Tabla: Json E/S usuario inicia sesión.....	26
Diagrama de flujo: usuario inicia sesión.....	26
Tabla: Json E/S usuario elimina su cuenta.....	27
Diagrama de flujo: usuario elimina su cuenta.....	27
Tabla: Json E/S usuario busca a más usuarios.....	28
Diagrama de flujo: usuario busca a más usuarios.....	28
Tabla: Json E/S usuario obtiene detalles de usuario.....	29
Diagrama de flujo: usuario obtiene detalles de usuario.....	29
Tabla: Json E/S usuario actualiza su cuenta.....	30
Diagrama de flujo: usuario actualiza su cuenta.....	30
Tabla: Json E/S usuario verifica si es administrador.....	31
Diagrama de flujo: usuario verifica si es administrador.....	31
Tabla: Json E/S usuario reto diario.....	32
Diagrama de flujo: usuario reto diario.....	32
Tabla: Json E/S usuario cambiar indumentaria.....	33

Diagrama de flujo: usuario cambiar indumentaria.....	33
Tabla: Rutina.....	34
Tabla: RutinaBuscadorDto.....	34
Tabla: RutinaDTO.....	35
Tabla: EjercicioDTO.....	35
Tabla: endpoints Rutina.....	36
Tabla: Json E/S usuario crea una rutina.....	37
Diagrama flujo: usuario crea una rutina.....	37
Tabla: Json E/S usuario ver rutinas.....	38
Diagrama flujo: usuario ver rutinas.....	38
Tabla: Json E/S usuario ver rutinas.....	39
Diagrama flujo: usuario ver rutinas.....	39
Tabla: Json E/S obtener detalles rutina.....	40
Diagrama flujo: obtener detalles de la rutina.....	40
Tabla: Json E/S usuario obtiene rutinas por autor.....	41
Diagrama flujo: usuario obtener rutinas por autor.....	41
Tabla: Json E/S usuario elimina una rutina.....	42
Diagrama flujo: usuario elimina una rutina.....	42
Tabla: Votos.....	43
Tabla: VotodDto.....	43
Tabla: endpoints Votos.....	44
Tabla: Json E/S registra voto.....	45
Diagrama flujo: registra voto.....	45
Tabla: Json E/S usuario obtiene votos.....	46
Diagrama flujo: usuario obtiene votos.....	46
Tabla: Json E/S actualiza su voto.....	47
Diagrama flujo: actualiza su voto.....	47
Tabla: Json E/S usuario se registra.....	48
Diagrama flujo: usuario se registra.....	48
Tabla: Json E/S usuario obtiene votos por autor.....	49
Diagrama flujo: usuario obtiene votos por autor.....	49
Tabla: PaymentRequestDto.....	50
Tabla: PaymentResponse.....	50
Tabla: endpoints Rutina.....	50
Tabla: Json E/S usuario crea intención de pago con Stripe.....	51
Diagrama flujo: usuario crea intención de pago con Stripe.....	51
Tabla: Json E/S Stripe webhook — manejo de notificaciones de pago.....	52
Diagrama flujo: Stripe webhook — manejo de notificaciones de pago.....	52
Tabla: Reporte.....	53
Tabla: UsuarioRegistroDTO.....	53
Tabla: endpoints Rutina.....	53
Tabla: Json E/S Reportar usuario.....	54
Diagrama flujo: Reportar usuario.....	54

Tabla: endpoints moderacion.....	55
Tabla: Json E/S verificar comentarios pendientes de moderación.....	56
Diagrama flujo: verificar comentarios pendientes de moderación.....	56
Tabla: Json E/S eliminar comentario por administrador.....	57
Diagrama flujo: eliminar comentario por administrador.....	57
Tabla: Json E/S obtener usuarios reportados.....	58
Diagrama flujo: Obtener usuarios reportados.....	58
Tabla: Json E/S Eliminar usuario reportado.....	59
Diagrama flujo: Eliminar usuario reportado.....	59
Tabla: EstadisticasDiarias.....	60
Tabla: EstadisticasDiariasDto.....	60
Tabla: endpoints Rutina.....	61
Tabla: Json E/S Obtener últimos 5 pesos registrados.....	61
Diagrama flujo: Obtener últimos 5 pesos registrados.....	61
Tabla: Json E/S Actualizar estadísticas diarias.....	62
Diagrama flujo: Actualizar estadísticas diarias.....	62
Tabla: Json E/S Obtener estadísticas diarias de un día.....	63
Diagrama flujo: Obtener estadísticas diarias de un día.....	63
Tabla: CoinPack.....	64
Tabla: endpoints coin-packs.....	64
Tabla: Json E/S usuario se registra.....	65
Diagrama flujo: usuario se registra.....	65
Tabla: Cosmetic.....	66
Tabla: endpoints Cosmetic.....	66
Tabla: Json E/S Obtener lista de cosméticos disponibles.....	67
Diagrama flujo: Obtener lista de cosméticos disponibles.....	67
Tabla: Compra.....	68
Tabla: endpoints compras.....	68
Tabla: Json E/S Obtener cosméticos comprados por el usuario.....	69
Diagrama flujo: Obtener cosméticos comprados por el usuario.....	69
Tabla: Json E/S Registrar una compra de cosmético.....	70
Diagrama flujo: Registrar una compra de cosmético.....	70
Tabla: Ejercicio.....	71
Tabla: EjercicioDTO.....	71
Tabla: endpoints ejercicios.....	72
Tabla: Json E/S Crear un ejercicio personalizado.....	73
Diagrama flujo: Crear un ejercicio personalizado.....	73
Tabla: Json E/S Obtener reto diario.....	74
Diagrama flujo: Obtener reto diario.....	74
Tabla: Json E/S Obtener lista de ejercicios.....	75
Diagrama flujo: Obtener lista de ejercicios.....	75
Tabla: Comentario.....	76
Tabla: ComentarioDto.....	77

Tabla: endpoints comentarios.....	78
Tabla: Json E/S Crear un comentario.....	79
Diagrama flujo: Crear un comentario.....	79
Tabla: Json E/S Obtener comentarios principales públicos.....	80
Diagrama flujo: Obtener comentarios principales públicos.....	80
Tabla: Obtener comentario principal y sus respuestas.....	81
Diagrama flujo: Obtener comentario principal y sus respuestas.....	81
Tabla: Json E/S Crear respuesta a un comentario.....	82
Diagrama flujo: Crear respuesta a un comentario.....	82
Tabla: Json E/S Eliminar comentario o respuesta.....	83
Diagrama flujo: Eliminar comentario o respuesta.....	83
Tabla: Json E/S Aprobar comentario.....	84
Diagrama flujo: Aprobar comentario.....	84
Tabla: Json E/S Obtener comentarios por autor.....	85
Diagrama flujo: Obtener comentarios por autor.....	85
Tabla: Json E/S Obtener comentarios por nombre de autor.....	86
Diagrama flujo: Obtener comentarios por nombre de autor.....	86
Tabla: Estadísticas.....	87
Tabla: EstadísticasDto.....	87
Tabla: endpoints Rutina.....	88
Tabla: Json E/S Obtener estadísticas de usuario.....	89
Diagrama flujo: Obtener estadísticas de usuario.....	89
Tabla: Json E/S Crear estadísticas de usuario.....	90
Diagrama flujo: Crear estadísticas de usuario.....	90
Tabla: Json E/S Actualizar estadísticas de usuario.....	91
Diagrama flujo: Actualizar estadísticas de usuario.....	91
Imágenes: Pruebas unitarias.....	93
Imagen: pantalla buscar rutinas.....	96
Imagen: pantalla crear rutina.....	96
Imagen: pantalla buscar rutinas.....	96
Imagen: pantalla detalles rutinas.....	96
Imagen: pantalla realizar rutina.....	96
Imagen: pantalla rutina fina.....	96
Imagen: pantalla mi zona.....	97
Imagen: pantalla reto diario.....	97
Imagen: pantalla estadísticas diarias.....	97
Imagen: pantalla equipar indumentaria.....	97
Imagen: pantalla estadistica anotar peso.....	97
Imagen: pantalla seleccionar foto avatar.....	97
Imagen: pantalla buscar comentario.....	98
Imagen: pantalla ver comentario.....	98
Imagen: pantalla eliminar comentario.....	98
Imagen: pantalla administrar comentarios con fotos.....	98

Imagen: pantalla administrar comentarios con fotos.....	98
Imagen: perfil de otro usuario.....	98
Imagen: pantalla tienda.....	99
Imagen: pantalla comprar cosmético.....	99
Imagen: pantalla tienda.....	99
Imagen: pantalla stripe.....	99
Imagen: pantalla stripe tarjeta.....	99
Imagen: pantalla pago correcto.....	99
Imagen: pantalla ajustes.....	100
Imagen: pantalla sin conexión.....	100
Imagen: pantalla cargando.....	100
Imagen: pantalla modo oscuro.....	100
Imagen: pantalla texto grande.....	100

## 13. Bibliografía

### 11.1 maquetación:

- <https://www.upv.es/entidades/BBAA/infoweb/fba/info/U0859586.pdf>

### 11.2 Mongodb:

- [\\$regex - Database Manual v8.0 - MongoDB Docs](#)

### 11.3 varios:

- [Navigation y la pila de actividades | App architecture | Android Developers](#)
- [Arquitectura de apps: Capa de datos - DataStore - Android Developers | App architecture](#)
- [Android DataStore: unresolved Reference edit, stringPreferencesKey - Stack Overflow](#)
- [DATASTORE PREFERENCES en ANDROID !\[\]\(bcda8d62fec25c4abcaf5e517e8c2f72\_img.jpg\) con KOTLIN DESDE CERO - \[ JETPACK en ANDROID STUDIO \]](#)
- [-Información acerca de la protección de la barra del sistema | Jetpack Compose | Android Developers](#)
- [Anuncios nativos | Android | Google for Developers](#)