



Facultad de Informática

Ingeniería en Software

Servicios Cloud

Alarcón Castillo Diego André -- 315427

Evaristo Camargo Daniel -- 315268

Pérez Rodríguez Sergio Alejandro



Documentación del Proyecto: Hunter Fitness

Fecha de Documentación: 2 de junio de 2025

Versión del Proyecto: v.1.0.6

Autores:

- Alarcón Castillo Diego André - 315427
 - Evaristo Camargo Daniel - 315268
-

1. Descripción General del Proyecto

Concepto Principal: Hunter Fitness es una aplicación móvil gamificada inspirada en el universo de "Solo Leveling" que transforma el entrenamiento físico en una experiencia RPG inmersiva. Los usuarios se convierten en "Hunters" que deben completar quests diarias de ejercicio para subir de nivel, desbloquear equipment y ascender en rankings globales.

Visión: Crear la aplicación de fitness más motivadora del mercado, donde cada usuario viva la experiencia de ser el protagonista de su propia historia de superación física, similar a Sung Jin-Woo en Solo Leveling.

Objetivos Principales:

- Gamificar completamente la experiencia de entrenamiento.
- Mantener engagement a través de mecánicas de progresión RPG.
- Crear comunidad mediante sistemas de guilds y rankings.
- Facilitar hábitos saludables a través de recompensas inmediatas.
- Escalabilidad para millones de usuarios.

Mecánicas de Juego Core:

- **Sistema de Levels y XP:** Niveles 1-100+ con curva de XP exponencial. Fuentes de XP: Daily Quests, Dungeons, Achievements.
 - **Hunter Stats System:** Strength, Agility, Vitality, Endurance.
 - **Ranking System:** Desde E-Rank hasta SSS-Rank.
 - **Daily Quests System:** 3-5 quests diarias adaptadas, tipos variados, dificultad escalable, multiplicadores de bonus.
 - **Dungeon Raids System:** Diferentes tipos de dungeons, requisitos de entrada, sistema de cooldown, recompensas altas.
 - **Equipment System:** Items (Weapons, Armor, Accessories), niveles de rareza, bonificaciones de stats, condiciones de desbloqueo.
 - **Achievement System:** Categorías variadas, logros progresivos y ocultos, recompensas (XP, Títulos, Equipment).
-



2. Arquitectura Técnica

El proyecto se compone de tres partes principales:

- **Frontend (Aplicación Móvil/Web):** Desarrollado con Flutter.
 - **Estructura:** Organizada en modelos, servicios, pantallas, widgets y utilidades.
 - **Plataformas Soportadas:** Android, iOS, Web (desplegado en Azure Static Web Apps).
- **Backend (API):** Desarrollado con Azure Functions (.NET 8).
 - **Estructura:** Organizada en Funciones HTTP (endpoints), Modelos, Servicios (lógica de negocio) y Helpers.
 - **Despliegue:** Azure Function App (hunter-fitness-api).
- **Base de Datos:** Azure SQL Database.
 - **Nombre del Servidor:** hunter-fitness-server.database.windows.net.
 - **Nombre de la Base de Datos:** HunterFitnessDB.

Recursos Azure Utilizados (Principales):

- **Azure Resource Group:** Hunter-Fitness-RG.
- **Azure SQL Database:** HunterFitnessDB
- **Azure Function App (hunter-fitness-api):**
- **Azure Static Web Apps (hunterfitness-app)** para el frontend web.

3. Diseño de la Base de Datos (Azure SQL Database)

El esquema de la base de datos está diseñado para soportar las mecánicas de juego y las funcionalidades de la aplicación. Las tablas principales se describen a continuación (basado en `hunter_fitness_schema.sql` y `query.sql`).

- **Hunters:** Almacena la información del perfil de los usuarios, incluyendo sus estadísticas, nivel, XP, rango y metadatos.
 - Clave Primaria: `HunterID` (`UNIQUEIDENTIFIER`).
 - Campos Únicos: `Username`, `Email`.
 - Constraint CHECK para `HunterRank`: ('E', 'D', 'C', 'B', 'A', 'S', 'SS', 'SSS').
- **DailyQuests:** Define las plantillas para las misiones diarias disponibles en el juego.
 - Clave Primaria: `QuestID` (`UNIQUEIDENTIFIER`).
 - Incluye tipo de quest, configuración del ejercicio, recompensas, dificultad y requisitos.
 - Constraint CHECK para `QuestType`: ('Cardio', 'Strength', 'Flexibility', 'Endurance', 'Mixed').
 - Constraint CHECK para `Difficulty`: ('Easy', 'Medium', 'Hard', 'Extreme').
- **HunterDailyQuests:** Tabla de unión que registra las misiones diarias asignadas a cada hunter, su progreso y estado.
 - Clave Primaria: `AssignmentID` (`UNIQUEIDENTIFIER`).
 - Claves Foráneas: `HunterID` (referencia a `Hunters`), `QuestID` (referencia a `DailyQuests`).
 - Constraint CHECK para `Status`: ('Assigned', 'InProgress', 'Completed', 'Failed').
- **Dungeons:** Define las mazmorras o raids disponibles, su dificultad, requisitos y recompensas.
 - Clave Primaria: `DungeonID` (`UNIQUEIDENTIFIER`).
 - Constraint CHECK para `DungeonType`: ('Training Grounds', 'Strength Trial', 'Endurance Test', 'Boss Raid').



UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

SOF-18, Semestre 6, IA



- Constraint CHECK para Difficulty: ('Normal', 'Hard', 'Extreme', 'Nightmare').
- **DungeonExercises:** Define los ejercicios específicos que componen cada dungeon.
 - Clave Primaria: ExerciseID (UNIQUEIDENTIFIER).
 - Clave Foránea: DungeonID (referencia a Dungeons).
- **DungeonRaids:** Registra los intentos de los hunters en los dungeons, su estado y resultados.
 - Clave Primaria: RaidID (UNIQUEIDENTIFIER).
 - Claves Foráneas: HunterID, DungeonID.
 - Constraint CHECK para Status: ('Started', 'InProgress', 'Completed', 'Failed', 'Abandoned').
- **Achievements:** Catálogo de logros disponibles en el sistema.
 - Clave Primaria: AchievementID (UNIQUEIDENTIFIER).
 - Constraint CHECK para Category: ('Consistency', 'Strength', 'Endurance', 'Social', 'Special', 'Milestone').
 - Constraint CHECK para AchievementType: ('Counter', 'Streak', 'Single', 'Progressive').
- **HunterAchievements:** Registra los logros desbloqueados por cada hunter y su progreso.
 - Clave Primaria: HunterAchievementID (UNIQUEIDENTIFIER).
 - Claves Foráneas: HunterID, AchievementID.
 - Constraint ÚNICA en (HunterID, AchievementID).
- **Equipment:** Catálogo del equipamiento disponible, sus bonificaciones y requisitos.
 - Clave Primaria: EquipmentID (UNIQUEIDENTIFIER).
 - Constraint CHECK para ItemType: ('Weapon', 'Armor', 'Accessory').
 - Constraint CHECK para Rarity: ('Common', 'Rare', 'Epic', 'Legendary', 'Mythic').
- **HunterEquipment:** Registra el equipamiento que posee cada hunter y si lo tiene equipado.
 - Clave Primaria: HunterEquipmentID (UNIQUEIDENTIFIER).
 - Claves Foráneas: HunterID, EquipmentID.
 - Constraint ÚNICA en (HunterID, EquipmentID).
- **QuestHistory:** Almacena un registro histórico de todas las misiones completadas por los hunters.
 - Clave Primaria: HistoryID (UNIQUEIDENTIFIER).
 - Claves Foráneas: HunterID, QuestID.
- **Guilds (Gremio/Grupos)** (definida en `hunter_fitness_schema.sql`): Información sobre los gremios creados por los hunters.
 - Clave Primaria: GuildID (UNIQUEIDENTIFIER).
 - Clave Única: GuildName.
- **GuildMembers (Miembros de Gremio)** (definida en `hunter_fitness_schema.sql`): Tabla de unión para los miembros de cada gremio.
 - Clave Primaria: MembershipID (UNIQUEIDENTIFIER).
 - Claves Foráneas: GuildID, HunterID.
 - Constraint ÚNICA en HunterID (un hunter solo puede estar en un guild).
 - Constraint CHECK para Role: ('Member', 'Officer', 'Leader').
- **LeaderboardCache (Cache de Rankings)** (definida en `hunter_fitness_schema.sql`): Almacena datos de ranking precalculados para optimizar consultas.
 - Clave Primaria: CacheID (UNIQUEIDENTIFIER).
 - Constraint CHECK para LeaderboardType: ('Global', 'Weekly', 'Monthly', 'Guild').

Índices Importantes: Se han creado varios índices para optimizar las consultas más frecuentes en la base de datos.

- En Hunters: IX_Hunters_Username, IX_Hunters_Email, IX_Hunters_Level, IX_Hunters_HunterRank.
- En HunterDailyQuests: IX_HunterDailyQuests_HunterID_Date, IX_HunterDailyQuests_Status.



- En DungeonRaids: IX_DungeonRaids_HunterID, IX_DungeonRaids_Status.
- En QuestHistory: IX_QuestHistory_HunterID_CompletedAt.
- En LeaderboardCache: IX_LeaderboardCache_Type_Rank.

Datos Semilla (query.sql): El archivo `query.sql` incluye sentencias `INSERT INTO` para poblar las tablas DailyQuests, Dungeons, DungeonExercises, Achievements, y Equipment con datos iniciales.

4. Componentes del Backend (Azure Functions API - `hunter_fitness_api`)

La API backend está construida utilizando Azure Functions con .NET 8 y se encarga de toda la lógica de negocio, autenticación y comunicación con la base de datos.

4.1. Configuración:

- **host.json:**
 - Define la configuración global para todas las funciones en la Function App.
 - Versión de funciones: "2.0".
 - **Logging:** Configurado para Application Insights con muestreo y filtros de métricas en vivo. Niveles de log por defecto en "Information", con advertencias para componentes de Microsoft.
 - **Extension Bundle:** Microsoft.Azure.Functions.ExtensionBundle, versión [4.*, 5.0.0).
 - **Timeout de Función:** "00:10:00".
 - **HTTP:**
 - Prefijo de ruta: "api".
 - **CORS:** Configurado para permitir todos los orígenes ("*") y métodos comunes (GET, POST, PUT, DELETE, OPTIONS) y cabeceras. Credenciales no permitidas.
 - **Retry Policy:** Estrategia de "exponentialBackoff" con hasta 3 reintentos.
- **appsettings.json:**
 - **Logging:** Niveles de log detallados.
 - **HunterFitness:** Sección principal de configuración de la aplicación.
 - ApiSettings: Nombre, versión, descripción de la API.
 - GameSettings: Parámetros del juego como nivel máximo, cantidad de quests, cooldowns.
 - SecuritySettings: Configuración para JWT (Issuer, Audience, expiración) y políticas de contraseña.
 - CacheSettings: Configuración de expiración para diferentes tipos de caché.
 - DatabaseSettings: Timeout de comandos, reintentos en caso de fallo.
 - **ConnectionStrings.DefaultConnection:** Cadena de conexión a la base de datos Azure SQL.
 - **ApplicationInsights:** Configuración para Application Insights.
- **Program.cs:**
 - Punto de entrada y configuración de servicios para la Function App.
 - Configura Application Insights Telemetry.
 - Configura el `DbContext (HunterFitnessDbContext)` con la cadena de conexión obtenida de variables de entorno o `appsettings.json`, y habilita reintentos en caso de fallo.
 - Registra los servicios de la aplicación con sus interfaces (Scoped): `IAuthService`, `IQuestService`, `IHunterService`, `IDungeonService`, `IEquipmentService`, `IAchievementService`.
 - Configura el logging, con niveles diferentes para Development y Production.



- Incluye un método `InitializeDatabaseAsync` para verificar la conexión con la base de datos al inicio y mostrar estadísticas básicas en desarrollo.

4.2. Modelos de Datos (C# - Carpeta Models): Estos archivos definen la estructura de las entidades de la base de datos y cómo se mapean a objetos C#. Cada modelo incluye propiedades, relaciones y métodos helper para lógica encapsulada.

- Achievement.cs
- DailyQuest.cs
- Dungeon.cs
- DungeonExercise.cs
- DungeonRaid.cs
- Equipment.cs
- Hunter.cs
- HunterAchievement.cs
- HunterDailyQuest.cs
- HunterEquipment.cs
- QuestHistory.cs

4.3. Data Transfer Objects (DTOs - Carpeta DTOs): Estos archivos definen las estructuras de datos que se utilizan para transferir información entre el cliente (aplicación Flutter) y la API, y entre diferentes capas de servicios en el backend.

- DungeonAndEquipmentDTOs.cs (Contiene DTOs para Equipment, HunterEquipment, Dungeon, DungeonExercise, DungeonRaid, Achievement, HunterAchievements y DTOs de respuesta genéricos como `ApiResponseDto` y `PaginatedResponseDto`).
- HunterDTOs.cs (Contiene DTOs para registro, login, perfil de hunter, resumen, actualización, stats, progreso, cambio de contraseña).
- QuestDTOs.cs (Contiene DTOs para misiones diarias asignadas, inicio/actualización/completado de quests, historial, estadísticas de quests, quests disponibles).

4.4. Servicios (Lógica de Negocio - Carpeta Services): Los servicios encapsulan la lógica de negocio principal de la aplicación.

- **AuthService.cs:**
 - Maneja el registro de nuevos hunters (`RegisterHunterAsync`).
 - Autentica a los usuarios (`LoginAsync`).
 - Genera y valida tokens JWT (`GenerateJwtToken`, `ValidateTokenAsync`, `GetHunterFromTokenAsync`).
 - Refresca tokens (`RefreshTokenAsync`).
 - Utiliza BCrypt.Net para el hashing de contraseñas.
 - Convierte la entidad `Hunter` a `HunterProfileDto`.
- **HunterService.cs:**
 - Obtiene el perfil del hunter (`GetHunterProfileAsync`).
 - Actualiza el perfil del hunter (`UpdateHunterProfileAsync`).
 - Maneja el cambio de contraseña (`ChangePasswordAsync`).
 - Obtiene el leaderboard de hunters (`GetLeaderboardAsync`).
 - Obtiene estadísticas detalladas del hunter (`GetHunterStatsAsync`).



UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

SOF-18, Semestre 6, IA



- Obtiene el progreso general del hunter (`GetHunterProgressAsync`).
- Maneja la adición de XP, actualización de stats, incremento de conteo de workouts y actualización de rachas.
- **QuestService.cs:**
 - Obtiene las misiones diarias asignadas a un hunter (`GetDailyQuestsAsync`).
 - Genera automáticamente misiones diarias si no existen para el día actual.
 - Permite iniciar una misión (`StartQuestAsync`).
 - Actualiza el progreso de una misión (`UpdateQuestProgressAsync`).
 - Completa una misión, otorga XP y actualiza el historial (`CompleteQuestAsync`).
 - Obtiene el historial de misiones completadas (`GetQuestHistoryAsync`).
 - Obtiene estadísticas agregadas sobre las misiones del hunter (`GetQuestStatsAsync`).
 - Lista las quests disponibles según el nivel del hunter (`GetAvailableQuestsAsync`).
- **DungeonService.cs:**
 - Obtiene los dungeons disponibles para un hunter (`GetAvailableDungeonsAsync`).
 - Obtiene los detalles de un dungeon específico (`GetDungeonDetailsAsync`).
 - Maneja el inicio de un raid en un dungeon (`StartRaidAsync`), verificando elegibilidad, cooldowns y raids activos.
 - Actualiza el progreso de un raid (`UpdateRaidProgressAsync`).
 - Completa un raid, otorgando recompensas si es exitoso (`CompleteRaidAsync`).
 - Permite abandonar un raid (`AbandonRaidAsync`).
 - Obtiene los raids activos y el historial de raids.
- **EquipmentService.cs:**
 - Obtiene el inventario completo de un hunter (`GetHunterInventoryAsync`).
 - Permite equipar un ítem, manejando la lógica de desequipar ítems del mismo tipo (`EquipItemAsync`).
 - Permite desequipar un ítem (`UnequipItemAsync`).
 - Lista todo el equipamiento disponible en el juego, indicando si el hunter lo posee o es elegible (`GetAvailableEquipmentAsync`).
 - Desbloquea equipamiento para un hunter (`UnlockEquipmentAsync`).
 - Obtiene los ítems actualmente equipados y estadísticas del inventario.
- **AchievementService.cs:**
 - Obtiene todos los logros de un hunter, incluyendo su progreso (`GetHunterAchievementsAsync`).
 - Lista los logros disponibles que el hunter aún no ha completado (`GetAvailableAchievementsAsync`).
 - Verifica y actualiza el progreso de los logros basado en eventos del juego (`CheckAndUpdateAchievementsAsync`).
 - Desbloquea un logro específico para un hunter (`UnlockAchievementAsync`).
 - Obtiene logros por categoría y estadísticas agregadas de logros.

4.5. Funciones HTTP (Endpoints de la API - Carpeta Functions): Estas clases exponen los métodos de los servicios como endpoints HTTP accesibles por la aplicación cliente.

- **AuthFunctions.cs:**
 - POST /api/auth/register: Registra un nuevo hunter.
 - POST /api/auth/login: Autentica a un hunter y devuelve un token JWT.
 - POST /api/auth/refresh: Refresca un token JWT existente.
 - GET /api/auth/validate: Valida un token JWT.
- **HunterFunctions.cs:**
 - GET /api/hunters/profile: Obtiene el perfil del hunter autenticado.



UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

SOF-18, Semestre 6, IA



- PUT /api/hunters/profile: Actualiza el perfil del hunter autenticado.
- GET /api/hunters/stats: Obtiene las estadísticas del hunter.
- GET /api/hunters/progress: Obtiene el progreso general del hunter.
- POST /api/hunters/change-password: Permite al hunter cambiar su contraseña.
- GET /api/hunters/leaderboard: Obtiene el ranking de hunters.
- POST /api/hunters/add-xp (Nivel de autorización Function): Endpoint interno para añadir XP.
- **QuestFunctions.cs:**
 - GET /api/quests/daily: Obtiene las misiones diarias del hunter.
 - POST /api/quests/start: Inicia una misión diaria.
 - PUT /api/quests/progress: Actualiza el progreso de una misión.
 - POST /api/quests/complete: Completa una misión diaria.
 - GET /api/quests/history: Obtiene el historial de misiones.
 - GET /api/quests/stats: Obtiene estadísticas de misiones.
 - GET /api/quests/available: Lista las quests disponibles.
 - POST /api/quests/generate: Genera misiones diarias para el hunter.
- **DungeonFunctions.cs:**
 - Endpoints para listar dungeons, ver detalles, iniciar/actualizar/completar/abandonar raids y ver historial de raids.
- **EquipmentFunctions.cs:**
 - Endpoints para gestionar el inventario del hunter, equipar/desequipar ítems, ver equipamiento disponible y estadísticas del inventario.
 - Endpoint interno (POST /api/equipment/unlock) para desbloquear equipamiento.
- **AchievementFunctions.cs:**
 - Endpoints para obtener los logros del hunter, logros disponibles, por categoría y estadísticas de logros.
 - Endpoints internos (POST /api/achievements/unlock, POST /api/achievements/check) para gestionar el desbloqueo y la verificación de logros.
- **HealthCheck.cs:**
 - GET /api/health: Endpoint para verificar la salud de la API, incluyendo la conexión a la base de datos y otros chequeos.
 - GET /api/ping: Endpoint simple para verificar que la API está respondiendo.

4.6. Contexto de la Base de Datos (`HunterFitnessDbContext.cs`):

- Define los `DbSet` para cada entidad, permitiendo a Entity Framework Core interactuar con la base de datos.
- Configura relaciones, claves foráneas, índices únicos y precisión de tipos decimales en el método `OnModelCreating`.

5. Componentes del Frontend (Aplicación Flutter - `hunter_fitness_app`)

La aplicación frontend está desarrollada en Flutter, permitiendo un desarrollo multiplataforma (iOS, Android, Web).

5.1. Estructura del Proyecto (General - `hunter_fitness_prompt.md`):

El frontend sigue una estructura organizada:

- `lib/models/`: Modelos de datos de la aplicación.
- `lib/services/`: Lógica para llamadas a API, almacenamiento local.



- `lib/screens/`: Pantallas principales de la interfaz de usuario.
- `lib/widgets/`: Componentes de UI reutilizables.
- `lib/utils/`: Ayudantes y constantes.
- `assets/`: Imágenes, animaciones, sonidos.

5.2. Configuración del Proyecto:

- `pubspec.yaml`:
 - Nombre de la aplicación: `hunter_fitness_app`.
 - Descripción: "Gamified fitness tracking app inspired by Solo Leveling".
 - Dependencias principales:
 - `flutter`: SDK de Flutter.
 - `http`: Para realizar solicitudes HTTP a la API.
 - `flutter_secure_storage`: Para almacenar de forma segura el token JWT.
 - `cupertino_icons`: Iconos de estilo iOS.
 - Dependencias de desarrollo: `flutter_test`, `flutter_lints`.
 - Assets: Define las rutas a imágenes, iconos, sonidos y animaciones.
- `main.dart`:
 - Punto de entrada de la aplicación Flutter.
 - Define el `MaterialApp`, el tema oscuro inspirado en Solo Leveling y la pantalla de inicio (`LoginScreen`).
- `analysis_options.yaml`:
 - Configura el analizador de Dart, incluyendo las reglas de lint recomendadas por `flutter_lints`.

5.3. Servicios del Frontend:

- `ApiService.dart`:
 - Encapsula toda la lógica de comunicación con la API backend (`hunter-fitness-api`).
 - `_baseUrl: "https://hunter-fitness-api.azurewebsites.net"`.
 - Utiliza `http` para las solicitudes y `flutter_secure_storage` para manejar el token JWT.
 - `_handleApiResponse()`: Método centralizado para procesar respuestas de la API, decodificar JSON y manejar errores comunes y la estructura de respuesta de la API (success, message, data, hunterProfile, token).
 - `login()`: Envía credenciales al endpoint `/auth/login`.
 - `registerUser()`: Envía datos de registro al endpoint `/auth/register`.
 - `getDailyQuests()`: Obtiene las misiones diarias del endpoint `/quests/daily`, incluyendo el token de autorización.
 - `logout()`: Elimina el token JWT del almacenamiento seguro.

5.4. Pantallas Principales (Screens - `lib/screens/`):

- `LoginScreen.dart`:
 - Permite a los usuarios ingresar su ID de Cazador/Email y contraseña.
 - Utiliza un `Form` con `TextField` para la entrada de datos y validación.
 - Llama a `ApiService.login()` para autenticar al usuario.
 - Maneja el estado de carga (`_isLoading`) y muestra mensajes de UI (`_uiMessage`).
 - Si el login es exitoso y se obtiene el perfil del cazador, navega a `HomeScreen`.
 - Proporciona un enlace para navegar a `RegistrationScreen`.



- **RegistrationScreen.dart:**
 - Permite a los nuevos usuarios registrarse proporcionando nombre de cazador, identificador único (usuario), email y contraseña.
 - Incluye validación para los campos del formulario y confirmación de contraseña.
 - Llama a `ApiService.registerUser()` para crear la nueva cuenta.
 - Si el registro es exitoso y se obtiene el perfil, navega a `HomeScreen`.
- **HomeScreen.dart:**
 - Pantalla principal después de un inicio de sesión exitoso. Recibe `hunterProfileData`.
 - Muestra la información del cazador (nombre, nivel, XP, rango) en una tarjeta de información.
 - Carga y muestra las misiones diarias del hunter llamando a `ApiService.getDailyQuests()`.
 - Maneja el estado de carga y los mensajes de error para las misiones.
 - Muestra un resumen de las estadísticas principales del cazador (Fuerza, Agilidad, Vitalidad, Resistencia).
 - Incluye botones para acciones rápidas (actualmente placeholders).
 - Permite cerrar sesión, lo que redirige a `LoginScreen`.
 - Utiliza `RefreshIndicator` para recargar las misiones diarias.

5.5. Configuración Específica de Plataformas (Web):

- **web/index.html:** Punto de entrada HTML para la aplicación web Flutter.
- **web/manifest.json:** Manifiesto de la aplicación web, utilizado para PWA y metadatos.
- **web/staticwebapp.config.json** (Añadido recientemente, basado en nuestras conversaciones):
 - Configura el `navigationFallback` para que las rutas de Flutter funcionen correctamente en Azure Static Web Apps.
 - Define `mimeTypes` para `.json` y `.webmanifest`.

(Se omiten los archivos de configuración específicos de Android, iOS, Windows, Linux y macOS que son mayormente generados por Flutter y no han sido el foco principal de las modificaciones recientes, a menos que se indique lo contrario).

6. Proceso de Despliegue

6.1. API Backend (`hunter-fitness-api`) en Azure Functions:

- La API se despliega en una Azure Function App.
- El despliegue se automatiza mediante un workflow de GitHub Actions definido en `.github/workflows/master_hunter-fitness-api.yml`.
- **Pasos del Workflow de la API:**
 1. Checkout del código.
 2. Configuración del entorno .NET (versión 8.0.x).
 3. Restauración de dependencias del proyecto C#.
 4. Compilación del proyecto C# en configuración Release.
 5. Publicación del proyecto C# (genera los artefactos para Azure Functions).
 6. Login a Azure usando credenciales de Service Principal almacenadas en secretos de GitHub.
 7. Despliegue a la Azure Function App (`hunter-fitness-api`) usando `Azure/functions-action@v1`.
- La URL base de la API es: <https://hunter-fitness-api.azurewebsites.net>



6.2. Aplicación Web Frontend (`hunter_fitness_app`) en Azure Static Web Apps:

- El frontend web de Flutter se despliega en Azure Static Web Apps (SWA).
- El despliegue se automatiza mediante un workflow de GitHub Actions definido en `.github/workflows/azure-static-web-apps-red-glacier-01554260f.yml`
- **Pasos del Workflow del Frontend SWA (modificado para Flutter):**
 1. Checkout del código.
 2. Configuración del entorno de Flutter usando `subosito/flutter-action@v2` (versión 3.29.2, canal stable).
 3. Ejecución de `flutter pub get` y `flutter build web --release` dentro del directorio `hunter_fitness_app`.
 4. Despliegue de los artefactos generados (desde `hunter_fitness_app/build/web`) a Azure Static Web Apps usando `Azure/static-web-apps-deploy@v1`.
- La URL de la aplicación web es: <https://red-glacier-01554260f.6.azurestaticapps.net>
- Se requiere un archivo `staticwebapp.config.json` en la carpeta `hunter_fitness_app/web/` para manejar el enrutamiento de Flutter.
- La configuración CORS de la API Backend (`hunter-fitness-api`) ha sido actualizada para permitir solicitudes desde la URL de la SWA.

7. Guía de Usuario (Funcionalidad Actual)

1. **Acceso a la Aplicación:**
 - Abrir la URL `https://red-glacier-01554260f.azurestaticapps.net` en un navegador web.
2. **Registro:**
 - Si es un nuevo usuario, hacer clic en el enlace de registro en la pantalla de login.
 - Completar el formulario con Nombre de Cazador, Identificador Único (usuario), Email y Contraseña.
 - Al registrarse exitosamente, el usuario es redirigido a la pantalla principal (`HomeScreen`).
3. **Inicio de Sesión (Login):**
 - Ingresar el Identificador Único (usuario) o Email y la Contraseña.
 - Al iniciar sesión exitosamente, el usuario es redirigido a la pantalla principal (`HomeScreen`).
4. **Pantalla Principal (`HomeScreen`):**
 - **Información del Cazador:** Muestra el nombre, nivel, XP actual/necesario y rango del cazador.
 - **Misiones Diarias:**
 - Lista las misiones diarias asignadas al cazador para el día actual.
 - Muestra el nombre, tipo (con ícono), descripción del objetivo, progreso y estado de cada misión.
 - Permite recargar las misiones tirando hacia abajo (RefreshIndicator).
 - Muestra un mensaje motivacional y un resumen del progreso de las misiones.
 - **Estado del Cazador:** Muestra las estadísticas base de Fuerza, Agilidad, Vitalidad y Resistencia.
 - **Acceso a Funciones:** Contiene botones para "Mis Misiones", "Explorar Mazmorras" y "Ver Perfil Completo" (actualmente son placeholders que muestran un Snackbar).
 - **Cerrar Sesión:** Un botón en la AppBar permite cerrar la sesión y volver a la pantalla de Login.



8. Trabajo Futuro y Roadmap (Basado en `hunter_fitness_prompt.md`)

Fase 1: MVP (Actual - Parcialmente Completado)

- Core hunter system (levels, stats, XP) - Implementado en backend, parcialmente visible en frontend.
- Daily quests con tipos de ejercicio básicos - Backend implementado, visualización básica en frontend.
- Logros básicos - Definidos en DB, lógica de backend y frontend por implementar.
- Sistema simple de equipamiento - Definido en DB, lógica de backend y frontend por implementar.
- Perfil y seguimiento de progreso - Implementado en backend, parcialmente visible en frontend.
- Leaderboard global - Backend implementado, frontend por implementar.

Fase 2: Enhanced Gameplay (Próximos Pasos)

- Sistema de Dungeon Raids.
- Categorías avanzadas de logros e integración completa.
- Sistema de rareza y mejora de equipamiento.
- Creación y gestión de Gremios (Guilds).
- Notificaciones Push.
- Funcionalidades sociales (amigos, compartir).

Fases Posteriores (3 y 4):

- Recomendaciones de entrenamiento con IA.
- Integración con dispositivos wearables.
- Tutoriales de ejercicios en video.
- Creación de entrenamientos personalizados.
- Funciones de suscripción premium.
- Optimización de rendimiento, localización, etc.

Modelo de Monetización (Planificado):

- Modelo Freemium con una suscripción "Premium Hunter License".
- Compras dentro de la aplicación (XP Boosters, cosméticos, energía extra).
- Asociaciones con gimnasios, marcas de fitness, etc.

Este documento proporciona una visión general del estado actual de Hunter Fitness. A medida que el proyecto evolucione, esta documentación será actualizada para reflejar nuevas funcionalidades, cambios en la arquitectura o ajustes en el diseño. Muchas gracias por su atención.

Anexo del GitHub: [AlarconCastilloDiegoAndre/HunterFitness: Aplicación Fitness basada en el anime exitoso Solo Leveling](https://github.com/AlarconCastilloDiegoAndre/HunterFitness)