

GYMPET

Trabajo de fin de grado



Marcos Calderón Y Rubén Montero

INDICE

1.Objetivos

2.Clases

3.Problemas

4.Originalidades

5.Mejoras a largo plazo

1. OBJETIVOS

Aplicación de Fitness con un Hámster como Símbolo de Marca

La aplicación de fitness tiene un hámster como el símbolo de marca dominante. El hámster es el avatar que se utiliza para representar el progreso del usuario. A medida que el usuario logra metas de fitness y nutrición, el hamster va avanzando físicamente dándole al usuario su propio logro. La gamificación y la motivación lograda mantienen a los usuarios comprometidos y entretenidos, lo que lleva a un entorno agradable y envolvente.

Sistema de Conteo Calórico y Base de Datos de Opciones Alimenticias

Descripción: Tiene una gran base de datos actualizada con varias opciones alimenticias. El usuario puede seleccionar algunas opciones alimenticias de la base de datos. El sistema muestra la información detallada relacionada con los ingredientes y las calorías de cada opción alimenticia. Una vez que el usuario selecciona su comida, la aplicación automáticamente suma las calorías consumidas y da un resumen del análisis nutricional del alimento. Esto permitiría al usuario monitorear muy de cerca sus calorías consumidas por día y, por lo tanto, planificar su dieta y alcanzar sus objetivos nutricionales deseados.

Creación y Manejo de Rutinas de Entrenamiento

Descripción: El sistema ofrece un sólido apoyo para la creación y gestión de rutinas de entrenamiento. Los usuarios pueden diseñar nuevas rutinas personalizadas, crear ejercicios específicos para ser incluidos en sus rutinas, así como elegir ejercicios de una gran variedad de ejercicios predefinidos. El sistema también es capaz de permitir a sus usuarios realizar ejercicios cronometrados y monitoreados. Con la ayuda de estas características, el usuario puede monitorear sus ejercicios a tiempo, recibir instrucciones precisas sobre los intervalos y el tiempo de descanso, y monitorear su progreso. Todas las características ayudan al usuario a estar en la mejor posición posible para realizar los ejercicios de la manera más eficiente para obtener el resultado máximo de su tiempo de ejercicio.

Desarrollo en Android Studio

Descripción: La aplicación se ha construido en Android Studio, la fuente oficial de Google para crear aplicaciones de Android. Esto significa que la aplicación está completamente diseñada junto con los dispositivos Android en consideración para que los usuarios disfruten de una experiencia de uso muy suave. Android Studio también está admitido por la integración de las capacidades y herramientas de vanguardia en el ecosistema Android, asegurando la compatibilidad y el rendimiento aceptable en una amplia variedad de dispositivos.

2.CLASES

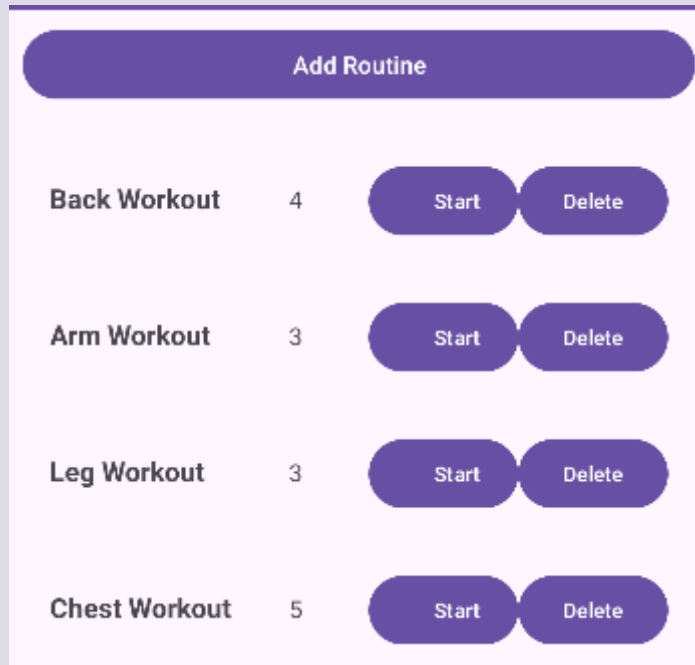
A) Routine

○ RoutineActivity

RoutineActivity es una clase en el código proporcionado que tiene la responsabilidad de mostrar y gestionar una rutina específica. A continuación, se presenta un resumen de RoutineActivity tomando como referencia la estructura proporcionada:

- Responsabilidad General:
Muestra los detalles de una rutina específica al usuario.
Permite la gestión de la rutina, como ver ejercicios, editar la rutina o eliminarla.
- Inicialización de Objetos:
Inicializa los elementos de la interfaz de usuario como TextView, RecyclerView, Button, y otros componentes necesarios para mostrar y gestionar la rutina.
Configura el adaptador del RecyclerView y los listeners para los componentes interactivos.
- Main:
Configura la interfaz de usuario al inflar el layout correspondiente.
Establece los listeners para manejar las interacciones del usuario, como clics en botones.
- Carga de Datos:
Carga los detalles de la rutina desde una base de datos local o una fuente de datos remota.
Actualiza el adaptador del RecyclerView con los ejercicios de la rutina.
- Gestión de Rutina:
Proporciona funcionalidades para ver detalles de la rutina, editarla o eliminarla.
Navega a otras actividades para la edición o visualización de detalles utilizando Intents.
- Manejo de Excepciones:
Captura y maneja cualquier excepción que pueda ocurrir durante la carga o gestión de datos, mostrando mensajes de error apropiados al usuario.

En resumen, RoutineActivity es responsable de mostrar y gestionar una rutina específica, permitiendo



al usuario ver detalles, editar o eliminar la rutina de manera eficiente.

○ **NewRoutineActivity**

NewRoutineActivity es una clase en el código proporcionado que tiene la responsabilidad de permitir al usuario crear una nueva rutina. A continuación, se presenta un resumen de NewRoutineActivity tomando como referencia la estructura proporcionada:

- **Responsabilidad General:**
Permite al usuario crear una nueva rutina.
Proporciona la interfaz de usuario y la lógica necesaria para capturar y guardar los detalles de la rutina.
- **Inicialización de Objetos:**
Inicializa los elementos de la interfaz de usuario como EditText, Button, y otros componentes necesarios para capturar la información de la rutina.
Configura los listeners para los botones y otros componentes interactivos.
- **Main:**
Configura la interfaz de usuario al inflar el layout correspondiente.

Establece los listeners para manejar las interacciones del usuario, como clics en botones.

- **Captura de Datos:**
Obtiene los datos ingresados por el usuario, como el nombre de la rutina y la descripción.
Valida que los datos sean correctos y estén completos antes de proceder.
- **Envío de Datos:**
Crea un objeto de rutina con los datos capturados.
Envía el objeto rutina de vuelta a la actividad llamante, utilizando un Intent.
- **Manejo de Excepciones:**
Captura y maneja cualquier excepción que pueda ocurrir durante la captura o envío de datos, mostrando mensajes de error apropiados al usuario.

En resumen, NewRoutineActivity es responsable de permitir al usuario crear una nueva rutina, capturando los datos necesarios y enviándolos de vuelta a la actividad principal de manera segura y eficiente.

○ **EditRoutineActivity**

EditRoutineActivity es una clase en el código proporcionado que tiene la responsabilidad de permitir al usuario editar una rutina existente. A continuación, se presenta un resumen de EditRoutineActivity tomando como referencia la estructura proporcionada:

- **Responsabilidad General:**
Permite al usuario editar una rutina existente.
Proporciona la interfaz de usuario y la lógica necesaria para modificar y guardar los detalles de la rutina.
- **Inicialización de Objetos:**
Inicializa los elementos de la interfaz de usuario como EditText, Button, y otros componentes necesarios para capturar la información modificada de la rutina.
Configura los listeners para los botones y otros componentes interactivos.
- **Main:**
Configura la interfaz de usuario al inflar el layout correspondiente.
Establece los listeners para manejar las interacciones del usuario, como clics en botones.

- **Captura de Datos:**
Obtiene los datos modificados por el usuario, como el nombre de la rutina y la descripción.
Valida que los datos sean correctos y estén completos antes de proceder.
- **Envío de Datos:**
Crea un objeto de rutina con los datos modificados.
Envía el objeto rutina de vuelta a la actividad llamante, utilizando un Intent.
- **Manejo de Excepciones:**
Captura y maneja cualquier excepción que pueda ocurrir durante la captura o envío de datos, mostrando mensajes de error apropiados al usuario.

En resumen, EditRoutineActivity es responsable de permitir al usuario editar una rutina existente, capturando los datos modificados y enviándolos de vuelta a la actividad principal de manera segura y eficiente.

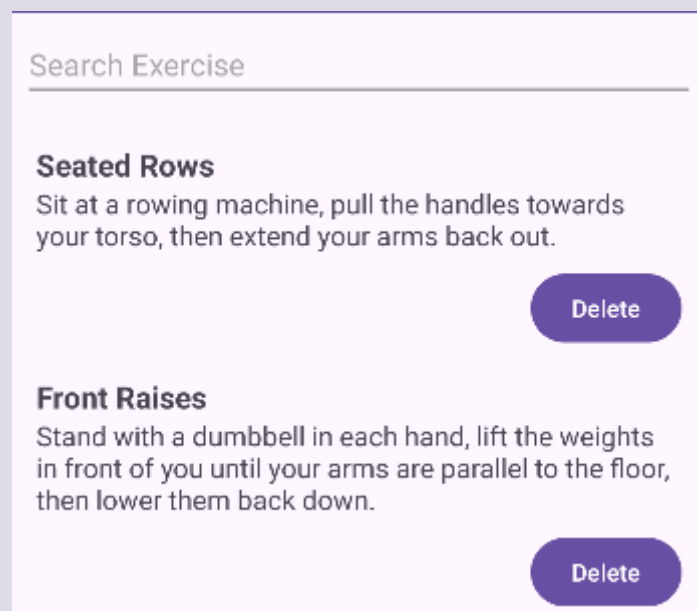
○ ExercisesActivity

ExercisesActivity es una clase en el código proporcionado que tiene la responsabilidad de mostrar y gestionar la lista de ejercicios disponibles. A continuación, se presenta un resumen de ExercisesActivity tomando como referencia la estructura proporcionada:

- **Responsabilidad General:**
Muestra una lista de ejercicios disponibles al usuario.
Permite la gestión de ejercicios, como ver detalles, editar o eliminar ejercicios.
- **Inicialización de Objetos:**
Inicializa los elementos de la interfaz de usuario como RecyclerView, FloatingActionButton, y otros componentes necesarios para mostrar y gestionar la lista de ejercicios.
Configura el adaptador del RecyclerView y los listeners para los componentes interactivos.
- **Main:**
Configura la interfaz de usuario al inflar el layout correspondiente.
Establece los listeners para manejar las interacciones del usuario, como clics en los elementos de la lista y el FloatingActionButton.
- **Carga de Datos:**
Carga la lista de ejercicios desde una base de datos local o una fuente de datos remota.
Actualiza el adaptador del RecyclerView con los datos cargados.

- **Gestión de Ejercicios:**
Proporciona funcionalidades para ver detalles de un ejercicio, editarlo o eliminarlo.
Navega a otras actividades para la edición o visualización de detalles utilizando Intents.
- **Manejo de Excepciones:**
Captura y maneja cualquier excepción que pueda ocurrir durante la carga o gestión de datos, mostrando mensajes de error apropiados al usuario.

En resumen, ExercisesActivity es responsable de mostrar y gestionar la lista de ejercicios disponibles, permitiendo al usuario ver detalles, editar o eliminar ejercicios de manera eficiente.



○ AddExerciseActivity

AddExerciseActivity es una clase en el código proporcionado que tiene la responsabilidad de permitir al usuario agregar un nuevo ejercicio a una rutina. A continuación, se presenta un resumen de AddExerciseActivity tomando como referencia la estructura proporcionada:

- **Responsabilidad General:**
Permite al usuario agregar un nuevo ejercicio a una rutina existente.
Proporciona la interfaz de usuario y la lógica necesaria para capturar y guardar los detalles del ejercicio.
- **Inicialización de Objetos:**
Inicializa los elementos de la interfaz de usuario como EditText, Button, y otros componentes

necesarios para capturar la información del ejercicio.

Configura los listeners para los botones y otros componentes interactivos.

- **Main:**
Configura la interfaz de usuario al inflar el layout correspondiente.
Establece los listeners para manejar las interacciones del usuario, como clics en botones.
- **Captura de Datos:**
Obtiene los datos ingresados por el usuario, como el nombre del ejercicio, la cantidad de repeticiones, series, etc.
Valida que los datos sean correctos y estén completos antes de proceder.
- **Envío de Datos:**
Crea un objeto de ejercicio con los datos capturados.
Envía el objeto ejercicio de vuelta a la actividad llamante, utilizando un Intent.
- **Manejo de Excepciones:**
Captura y maneja cualquier excepción que pueda ocurrir durante la captura o envío de datos, mostrando mensajes de error apropiados al usuario.

En resumen, AddExerciseActivity es responsable de permitir al usuario agregar un nuevo ejercicio a una rutina, capturando los datos necesarios y enviándolos de vuelta a la actividad principal de manera segura y eficiente.

○ **StartRoutineActivity**

StartRoutineActivity es una clase en el código proporcionado que tiene la responsabilidad de permitir al usuario iniciar una rutina. A continuación, se presenta un resumen de StartRoutineActivity tomando como referencia la estructura proporcionada:

- **Responsabilidad General:**
Permite al usuario iniciar una rutina.
Proporciona la interfaz de usuario y la lógica necesaria para seguir y completar los ejercicios de la rutina.
- **Inicialización de Objetos:**
Inicializa los elementos de la interfaz de usuario como TextView, Button, y otros componentes necesarios para guiar al usuario a través de los ejercicios.

Configura los listeners para los botones y otros componentes interactivos.

- **Main:**
Configura la interfaz de usuario al inflar el layout correspondiente.
Establece los listeners para manejar las interacciones del usuario, como clics en botones.
- **Seguimiento de Ejercicios:**
Muestra el ejercicio actual al usuario y permite la navegación a través de los ejercicios de la rutina.
Proporciona funcionalidad para marcar ejercicios como completados y avanzar al siguiente.
- **Envío de Datos:**
Guarda el progreso de la rutina y envía los datos de vuelta a la actividad llamante, utilizando un Intent.
- **Manejo de Excepciones:**
Captura y maneja cualquier excepción que pueda ocurrir durante el seguimiento o envío de datos, mostrando mensajes de error apropiados al usuario.

En resumen, `StartRoutineActivity` es responsable de permitir al usuario iniciar y seguir una rutina, capturando el progreso y enviándolo de vuelta a la actividad principal de manera segura y eficiente.

○ **EndRoutineActivity**

`EndRoutineActivity` es una clase en el código proporcionado que tiene la responsabilidad de permitir al usuario finalizar una rutina. A continuación, se presenta un resumen de `EndRoutineActivity` tomando como referencia la estructura proporcionada:

- **Responsabilidad General:**
Permite al usuario finalizar una rutina.
Proporciona la interfaz de usuario y la lógica necesaria para capturar y guardar el resultado de la rutina.
- **Inicialización de Objetos:**
Inicializa los elementos de la interfaz de usuario como `TextView`, `Button`, y otros componentes necesarios para mostrar el resumen de la rutina.
Configura los listeners para los botones y otros componentes interactivos.
- **Main:**
Configura la interfaz de usuario al inflar el layout correspondiente.

Establece los listeners para manejar las interacciones del usuario, como clics en botones.

- **Captura de Resultados:**
Obtiene los resultados de la rutina, como el tiempo total, repeticiones completadas, etc.
Valida que los datos sean correctos y estén completos antes de proceder.
- **Envío de Resultados:**
Crea un objeto con los resultados de la rutina.
Envía el objeto resultados de vuelta a la actividad llamante, utilizando un Intent.
- **Manejo de Excepciones:**
Captura y maneja cualquier excepción que pueda ocurrir durante la captura o envío de datos, mostrando mensajes de error apropiados al usuario.

En resumen, EndRoutineActivity es responsable de permitir al usuario finalizar una rutina, capturando los resultados y enviándolos de vuelta a la actividad principal de manera segura y eficiente.

○ **Clases secundarias**

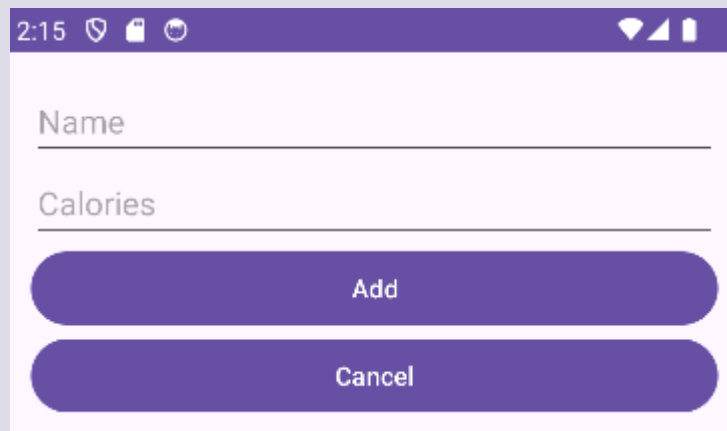
- **ExerciseAdapter**
ExerciseAdapter es un adaptador para el RecyclerView que muestra una lista de ejercicios en una interfaz de usuario.
- **ExerciseWithSets**
ExerciseWithSets es una clase que agrupa un ejercicio con sus respectivos sets (series) para facilitar su manejo conjunto.
- **Routine**
Routine es una clase que representa una rutina de ejercicios, incluyendo detalles como el nombre y la lista de ejercicios asociados.
- **RoutineAdapter**
RoutineAdapter es un adaptador para el RecyclerView que muestra una lista de rutinas en una interfaz de usuario.
- **TrainingAdapter**
TrainingAdapter es un adaptador para el RecyclerView que muestra una lista de entrenamientos, cada uno con sus detalles correspondientes.
- **EditRoutineAdapter**
EditRoutineAdapter es un adaptador para el RecyclerView que permite la edición de los ejercicios dentro de una rutina.
- **Exercise**
Exercise es una clase que representa un ejercicio individual, incluyendo detalles como el nombre y las repeticiones.

- **ExerciseDesc**
ExerciseDesc es una clase que proporciona una descripción detallada de un ejercicio, incluyendo instrucciones y consejos.
- **ExerciseDescAdapter**
ExerciseDescAdapter es un adaptador para el RecyclerView que muestra las descripciones detalladas de los ejercicios.
- **RoutineExerciseAdapter**
RoutineExerciseAdapter es un adaptador para el RecyclerView que muestra los ejercicios dentro de una rutina específica, permitiendo la interacción con ellos.

B) Nutrition

• **FoodAdapter.kt**

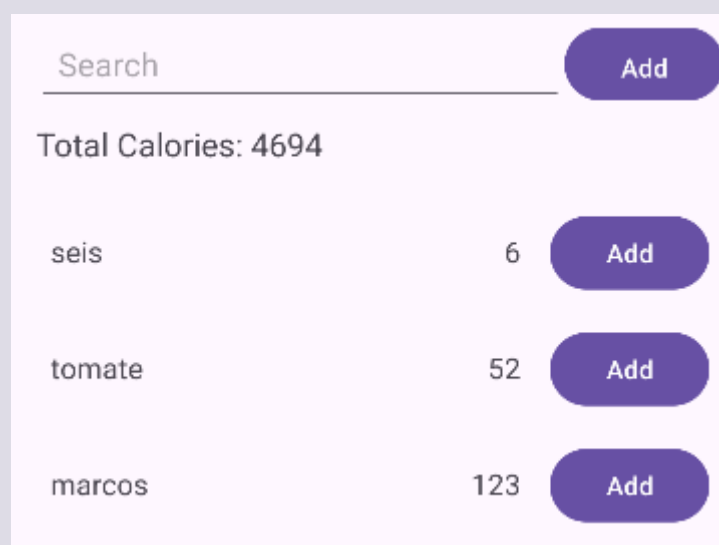
- Esta clase es un adaptador para mostrar elementos de comida en un RecyclerView.
 - **Propósito:** Vincular elementos de comida a un RecyclerView y gestionar su visualización e interacciones.
 - **Componentes Clave:**
 - **foodItems:** Una lista de elementos de comida a mostrar.
 - **filteredFoodItems:** Una lista filtrada de elementos de comida basada en consultas de búsqueda.
 - **onCreateViewHolder():** Infla el diseño para elementos de comida individuales.
 - **onBindViewHolder():** Vincula los datos del elemento de comida al view holder.
 - **updateItems():** Actualiza la lista de elementos de comida.
 - **filter():** Filtra la lista basada en una consulta de búsqueda.
 - **updateItem():** Actualiza un elemento de comida específico.
 - **FoodViewHolder:** Clase interna que contiene referencias a las vistas para cada elemento.



A screenshot of a mobile application interface. At the top, there is a status bar with the time 2:15 and various icons. Below the status bar, there is a form with two input fields: "Name" and "Calories". Each field has a horizontal line below it for text entry. At the bottom of the form, there are two large, rounded rectangular buttons: "Add" and "Cancel".

- **FoodItem.kt**

- Esta clase representa un elemento de comida.
 - **Propósito:** Definir la estructura de un elemento de comida.
 - **Componentes Clave:**
 - id: Un identificador único para el elemento de comida.
 - name: El nombre del elemento de comida.
 - calories: La cantidad de calorías del elemento de comida.
 - isSelected: Un booleano que indica si el elemento está seleccionado.



A screenshot of a mobile application interface showing a list of food items. At the top, there is a search bar with the placeholder text "Search" and an "Add" button. Below the search bar, the text "Total Calories: 4694" is displayed. The list contains three items, each with a name, a value, and an "Add" button:

Item Name	Value	Action
seis	6	Add
tomate	52	Add
marcos	123	Add

- **NutritionScreen.kt**

- Esta clase representa una pantalla en la aplicación que muestra información relacionada con la nutrición.
 - **Propósito:** Mostrar la pantalla de nutrición y manejar las interacciones del usuario con ella.
 - **Componentes Clave:**
 - `onCreate()`: Inicializa la pantalla y configura el RecyclerView con FoodAdapter.
 - `loadFoodItemsFromFirestore()`: Carga elementos de comida desde Firebase Firestore.
 - `filterFoodItems()`: Filtra los elementos de comida según la entrada del usuario.

- **AddedFoodAdapter.kt**

- Esta clase es otro adaptador, presumiblemente para mostrar una lista de elementos de comida que han sido añadidos por el usuario.
 - **Propósito:** Gestionar la visualización e interacciones de los elementos de comida añadidos en un RecyclerView.
 - **Componentes Clave:**
 - Estructura y funciones similares a FoodAdapter, adaptadas para los elementos de comida añadidos.



- **5. AddFoodActivity.kt**

- Esta clase representa una actividad donde los usuarios pueden añadir nuevos elementos de comida.
 - **Propósito:** Permitir a los usuarios ingresar y guardar nuevos elementos de comida en la base de datos.
 - **Componentes Clave:**
 - nameEditText y caloriesEditText: Campos de entrada para el nombre y las calorías del elemento de comida.
 - addButton y cancelButton: Botones para añadir el elemento de comida o cancelar la acción.
 - addFoodToFirestore(): Guarda el nuevo elemento de comida en Firebase Firestore.

C) MainActivity

Esta clase representa la actividad principal de la aplicación.

- **Propósito:** Actuar como la pantalla principal de la aplicación, gestionando la interfaz de usuario y la navegación entre diferentes secciones.

Componentes Clave:

- **Views:**
 - totalCaloriesTextView: Un TextView que muestra el total de calorías.
 - progressBar: Un ProgressBar que indica el progreso de puntos de entrenamiento.
 - imageView: Un ImageView que muestra una imagen de mascota.
 - buttonAddPoints y buttonSubtractPoints: Botones para agregar o restar puntos de entrenamiento.
 - imageViewInfo: Un ImageView que muestra información adicional.
 - buttonShowImage: Un botón para mostrar/ocultar la imagen de información.
- **SharedPreferences:**
 - sharedPreferences: Almacena los datos persistentes como calorías totales, progreso y nivel de la mascota.

Funcionalidad Principal:

- **onCreate():** Inicializa la actividad, establece el diseño y configura los elementos de la interfaz de usuario.
 - Carga y muestra las calorías totales desde SharedPreferences.
 - Configura los botones para agregar y restar puntos de entrenamiento.


- Configura el botón para mostrar/ocultar la imagen de información.
- **updateTotalCaloriesTextView(calories: Int):** Actualiza el texto del TextView de calorías totales.
- **addTrainingPoints():** Agrega puntos de entrenamiento, actualiza la barra de progreso y maneja el nivel de la mascota.
 - Incrementa el progreso en 25.
 - Si el progreso alcanza 100, reinicia a 0 y sube el nivel de la mascota.
 - Guarda el progreso en SharedPreferences.
- **subtractTrainingPoints():** Resta puntos de entrenamiento, actualiza la barra de progreso y maneja el nivel de la mascota.
 - Decrementa el progreso en 25.
 - Si el progreso es menor que 0, lo establece en 75 y baja el nivel de la mascota.
 - Guarda el progreso en SharedPreferences.
- **levelUp():** Sube el nivel de la mascota y actualiza la imagen correspondiente si el nivel es menor que 5.
- **levelDown():** Baja el nivel de la mascota y actualiza la imagen correspondiente si el nivel es mayor que 0.
- **updateMascotalImage():** Actualiza la imagen de la mascota según el nivel actual.

Esta actividad principal proporciona una interfaz interactiva donde los usuarios pueden ver y gestionar su progreso de entrenamiento y niveles de mascota, junto con la visualización de calorías totales.

3. Problemas encontrados:

En el desarrollo de la aplicación, nos hemos encontrado con diferentes problemas y contratiempos, y solo ha habido uno que no hemos podido solucionar.

Para la lista de elementos de comida, y de ejercicios y rutinas, lo primero que se nos vino a la mente fue utilizar API.

Platform API Editions API Demo Developers About Contact

Sign InStart Free

FatSecret Platform API Documentation

Welcome to the FatSecret Platform REST API

We provide a REST-like API that allows developers to integrate FatSecret Platform features into their applications. To get started:

1. [Register](#) for an account.
2. Generate an application Consumer Key and associated Consumer Secret.

All Requests to the FatSecret Platform REST API should be directed to "<https://platform.fatsecret.com/rest/server.api>".

Requests must include a method parameter, distinguishing the different types of calls available. For example, a correctly authenticated request to the food.get.v3 API method would appear as follows:

OAuth2.0 | OAuth1.0

```
POST https://platform.fatsecret.com/rest/server.api
Content-Type: application/json
Header: Authorization: Bearer {Access Token}
Parameters: method=food.get.v2&food_id=336918&format=json
```

Please take some time to read our [authentication guide](#). OAuth2.0 and OAuth1.0 are supported.

Guides

Welcome

Authentication

OAuth 2.0

OAuth 1.0

3 Legged OAuth

Data Types

Parameters

Storable Data

Error Codes

Localization*

Libraries

Foods

Food Brands

Food Categories

Food Sub Categories

Recipes

Recipe Types

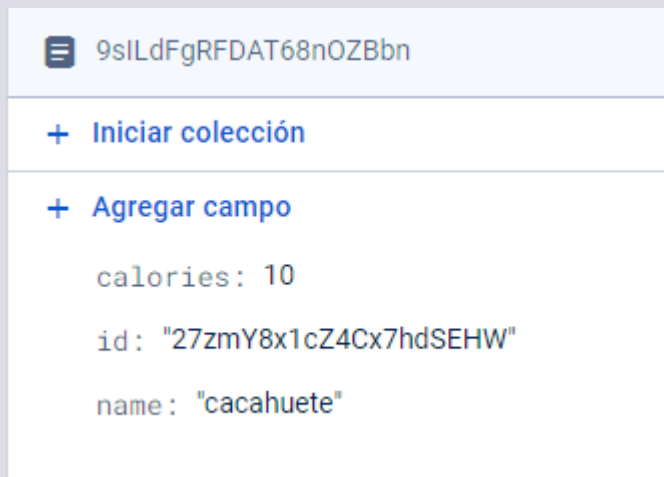
Web de [FatSecret](#), una de las paginas de nutrición mas famosas en españa

El problema, es que la inmensa mayoría son de pago, por lo que hemos tenido que optar por usar [firebase](#) en su lugar.

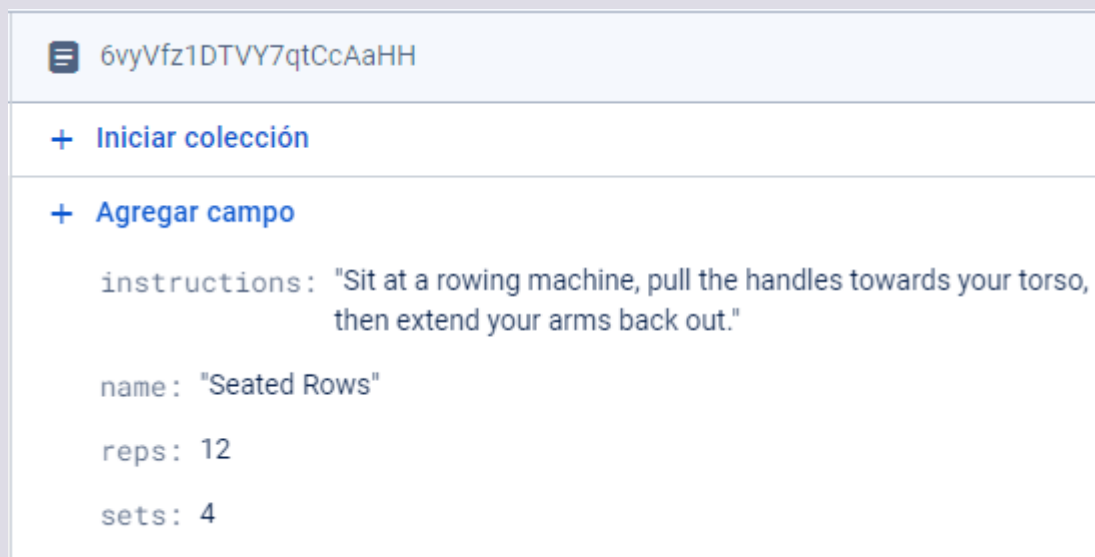


4. Originalidades

- Utilización de firestore como método de guardado de datos, usando ficheros de javascript para introducir varios campos en masa:

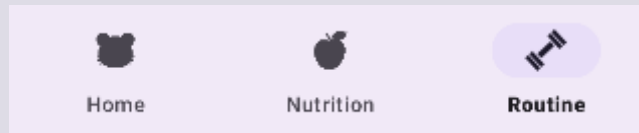


Colección de alimentos

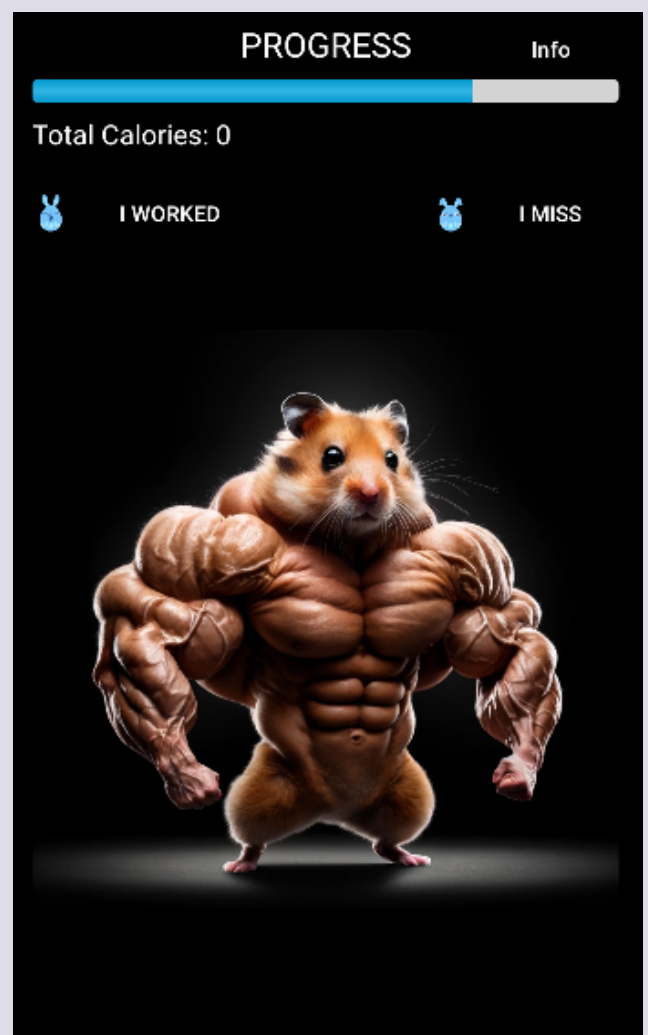
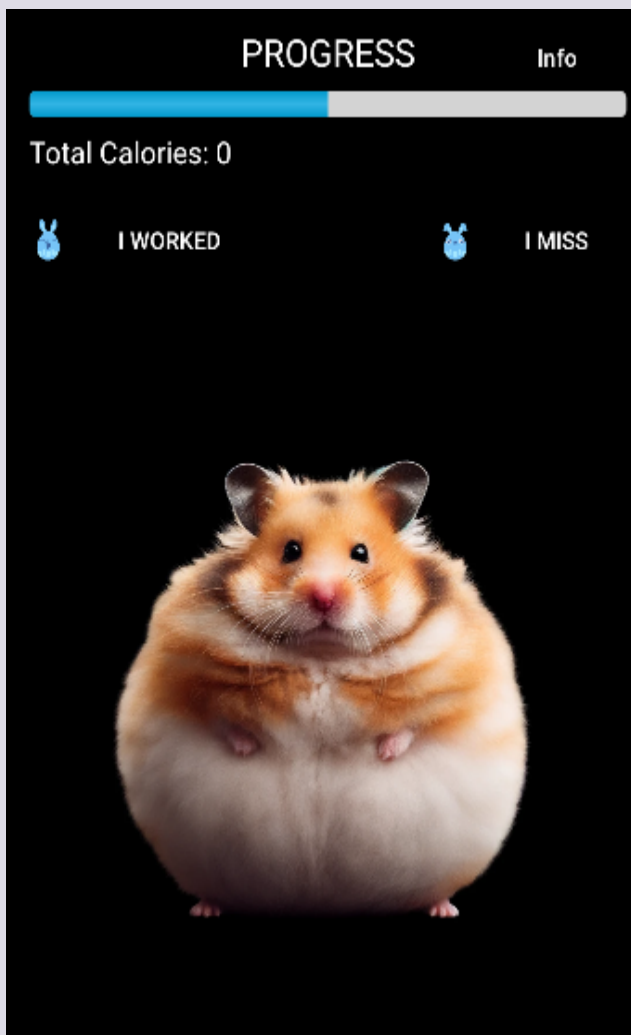


Colección de ejercicios

Navigation bar para ir de pantalla en pantalla



- *Progress Bar para mostrar el progreso del usuario:*



5. Mejoras a largo plazo

Esta aplicación tiene mucho potencial, ya que se trata de un nuevo enfoque al fitness, usando una mascota para representar el progreso del usuario. Hay 3 enfoques que se le podrían dar al proyecto en caso de que siguiera adelante:

1. Integración con Dispositivos y Aplicaciones de Salud

Permitir la sincronización de la aplicación con dispositivos portátiles de fitness (como Fitbit, Apple Watch, Garmin) y otras aplicaciones de salud (como Google Fit, Apple Health).

Beneficio Comercial: Mejorar la retención de usuarios ofreciendo una experiencia más integrada y personalizada, facilitando el seguimiento de la actividad física y la nutrición en un solo lugar. Esto puede atraer a un público más amplio y fidelizar a los usuarios existentes.

2. Plan de Nutrición Personalizado y Asesoramiento en Tiempo Real

Implementar una funcionalidad que ofrezca planes de nutrición personalizados basados en los objetivos y preferencias de cada usuario, con la posibilidad de recibir asesoramiento en tiempo real de nutricionistas y entrenadores a través de la app.

Beneficio Comercial: Ofrecer un valor añadido que diferencie la aplicación de otras en el mercado, aumentando las posibilidades de monetización a través de suscripciones premium o servicios adicionales. Además, esto puede aumentar la satisfacción y el éxito de los usuarios en alcanzar sus objetivos de fitness y nutrición.

3. Gamificación Avanzada y Recompensas

Ampliar la gamificación existente con más niveles, retos, y recompensas tangibles (como descuentos en productos de fitness, acceso a contenido exclusivo, etc.). Incluir funcionalidades sociales para compartir logros y competir con amigos.

Beneficio Comercial: Aumentar la motivación y el compromiso de los usuarios a largo plazo, promoviendo un uso más frecuente de la aplicación. La posibilidad de obtener recompensas puede incentivar a los usuarios a comprar productos o servicios asociados, generando ingresos adicionales.