DuocUC

Documentación Backend – Kachate utilizando Firebase y Node.js

Integrantes:

Luis Hinojosa

Damián Henríquez

Francisco Arriagada

Docente a Cargo:

Arturo Alex Vargas Reyes

Sección 001D

DuocUC

Contenido

1.Introducción 3

2. Stack tecnológico 4

Lenguaje y entorno de ejecución 4

Framework principal 4

Servicios de nube y base de datos 5

3. Arquitectura y flujo general 6

Flujo de acción Admin 6

Envió de la petición al backend con ID Token Firebase 7

Validación del ID token 7

Verificación del rol (¿Es administrador?) 7

Selección de la acción administrativa 7

Ejecución de la acción seleccionada 8

Respuesta final al administrador 9

Final del proceso 9

4.Requisitos para ejecutar en local 9

5.Endpoints (REST) – rutas principales 9

Endpoint de Estado de servicio 9

Endpoint análisis de ingredientes (OCR) 10

Endpoint Historial de usuario 11

Endpoints de administración 13

Reporte global admin 13

Crear ingrediente 13

Actualizar ingrediente 15

Eliminar ingredientes 16

Listar ingredientes 16

6.Conclusion 17

1.Introducción

El backend de Kachate constituye la parte lógica y de gestión de datos de la aplicación móvil orientada a la nutrición inteligente. Su propósito principal es proporcionar una capa de servicios que permita administrar usuarios, alimentos, roles y datos provenientes del reconocimiento óptico de caracteres (OCR) obtenido desde la aplicación Kachate para la plataforma de Android. De esta forma, se logra una comunicación segura, eficiente y escalable entre la aplicación móvil y la base de datos en la nube.

El backend fue diseñado en un enfoque modular, aplicando principios de arquitectura limpia para garantizar la mantenibilidad y facilidad de extensión futura (Como agregar más tipos de dietas). Utiliza el framework Express.js sobre Node.js, y se integra directamente con los servicios de Firebase, principalmente Firebase (Para la persistencia de datos) y Firebase Auth para el control de acceso y validación de usuarios.

En términos funcionales, el backend permite:

- Procesar solicitudes RESTful provenientes de la aplicación móvil Kachate
- Validar la identidad y rol de cada usuario
- Registrar, actualizar y consultar información nutricional, tanto manual como proveniente de OCR.
- Administrar permisos de acceso de acuerdo con los roles definidos (usuario y administrador)
- Facilitar el flujo de datos hacia los módulos de análisis y recomendaciones de alimentos dentro del ecosistema Kachate.

En el ámbito técnico, se prioriza la seguridad de la información mediante el uso de HTTPS, validaciones del lado servidor y la gestión de credenciales a través de entornos controlados. Asimismo, el uso de los Firebase Emulators en desarrollo y Firebase Functions en despliegue permite un entorno consistente, económico y con integración nativa en la nube.

Este documento describe de manera detallada los componentes técnicos del backend, las rutas disponibles, los procedimientos de despliegue, así como las buenas prácticas y lineamientos adoptados por el equipo de desarrollo.

2. Stack tecnológico

El desarrollo del backend de Kachate se base en una arquitectura moderna orientada a servicios, utilizando herramientas ampliamente adoptadas en la industria. A continuación, se detalla el conjunto de tecnologías, librerías y servicios empleados, junto con su función dentro del sistema.

Lenguaje y entorno de ejecución

	Node.js (Versión 20): proporciona el entorno de ejecución del servidor, permitiendo el manejo del servidor, permitiendo el manejo eficiente de operaciones asíncronas y la comunicación HTTP
	JavaScript: Lenguaje base para la implementación del código fuente, facilitando la compatibilidad con el ecosistema Firebase y las dependencias del proyecto.

Framework principal

	Express.js: Framework minimalista para Node.js que permite definir rutas, controladores y middlewares, asegurando una estructura modular y escalable para la API

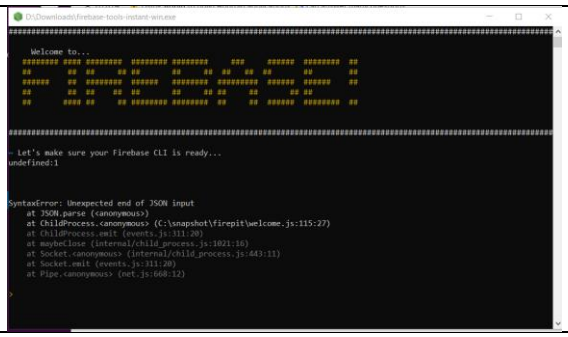
DuocUC

Servicios de nube y base de datos

	Firebase Firestore: base de datos no SQL en la nube que almacena información de usuarios, alimentos y registros de actividad. Se caracteriza por su alta disponibilidad y sincronización en tiempo real
	Firebase Authentication: Sistema de autenticación que gestiona el registro, inicio de sesión y validación de tokens de los usuarios.
	Firebase Emulators: Herramienta local que permite simular un entorno de producción para pruebas, evitando costos y facilitando el desarrollo colaborativo
	Firebase Functions (2° generación): Plataforma serverless utilizada para desplegar la API, eliminando la necesidad de un servidor físico o un contenedor manual

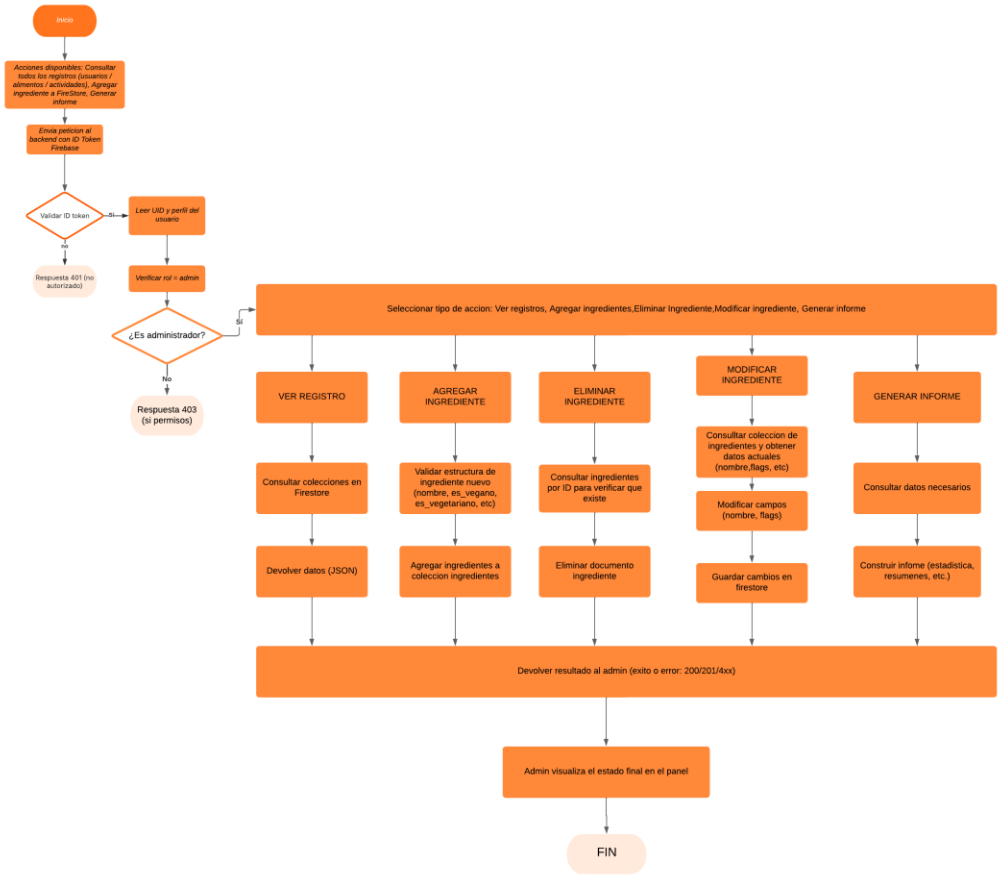
Herramientas de desarrollo y gestión

	NPM: Encargado de la instalación y administración de dependencias del proyecto

	Firebase CLI: Interfaz de línea de comandos para gestionar emuladores, funciones y despliegue.
GitHub	Git y GitHub: Control de versiones y colaboración entre los miembros del equipo de desarrollo

3. Arquitectura y flujo general

Flujo de acción Admin



Envió de la petición al backend con ID Token Firebase

La aplicación cliente envía una petición HTTP hacia el backend de Kachate incluyendo:

- El ID Token de Firebase authentication

Validación del ID token

El backend valida el token usando Firebase Admin SDK

- Si el token no es valido
    - Se responde 401 (No autorizado)
- Si es valido
    - Se continua con la lectura del usuario

Verificación del rol (¿Es administrador?)

El backend comprueba si el usuario es administrador.

- Si “NO” es administrador
    - Devuelve 403 (Sin permisos)
- Si “SI” es administrador
    - Puede acceder a las herramientas administrativas

A partir de este punto se confirma que el usuario:

- Tiene identidad valida
- Existe en el Firestore
- Tiene el rol de “admin”
- Puede ejecutar operaciones administrativas

Selección de la acción administrativa

Una vez que se valida el rol, el backend procesa la acción solicitada.

Las acciones son:

- Ver registros
- Agregar ingrediente
- Eliminar ingrediente
- Modificar ingrediente
- Generar informe

Ejecución de la acción seleccionada

A. Ver registros
   - Consultar colecciones en Firestore
     - Usuarios
     - Ingredientes clasificados
     - Historial de escaneos
   - Devolver los resultados en formato JSON

B. Agregar ingrediente
   - Validar estructura del ingrediente nuevo (nombre, es_vegano, es_vegetariano, es_celiaco, etc.)
   - Insertar el nuevo ingrediente en la colección **“ingredientes”**
   - Confirmar la creación con un código 201 – Creado

C. Eliminar ingrediente
   - Consultar el ingrediente por ID para verificar que existe
   - Eliminar el documento correspondiente
   - Devolver 204 – Sin contexto o un mensaje de confirmación

D. Modificar ingrediente
   - Consultar el ingrediente actual por su ID
   - Actualizar sus propiedades (nombre, flags nutricionales, descripción)
   - Guardar cambios en Firestore
   - Devolver 200 – Modificado con éxito

E. Generar informe
   - Consultar colecciones relevantes
   - Extraer datos estadísticos (número de ingredientes, escaneos, usuarios activos, etc.)
   - Construir un informe con estadísticas, resúmenes y tendencias
   - Devolver el informe en un formato PDF u otro definido

DuocUC

Respuesta final al administrador

Independiente de la acción ejecutada, el backend devolverá un resultado adecuado:

- 200 – OK
- 201 – Creado
- 204 – Sin contenido
- 4xx – Errores (datos inválidos, inexistencia, permisos)

El panel administrativo muestra resultado con una notificación toast.

Final del proceso

El administrador visualiza el resultado final y puede continuar con otras acciones dentro del sistema.

4.Requisitos para ejecutar en local

- Node.js >= 20
- Firebase CLI (NPM i -g Firebase-tools)
- Cuenta de Firebase con Firebase y Auth habilitados
- NPM instalado
- Proyecto Firebase inicializado con:
    - Firestore
    - Authentication
    - Functions (Node.js 20)
- Permisos de escritura en carpeta del proyecto
- Haber realizado un **Firebase login** en el proyecto

5.Endpoints (REST) – rutas principales

En esta sección se documentan los endpoints expuestos por el backend de Kachate, junto con los requisitos necesarios para su funcionamiento.

Endpoint de Estado de servicio

Método GET

Este Endpoint nos permite verificar si el backend está activo

Requisitos: No necesita autorización

Respuesta

```
{“status": "Backend Kachate activo”}
```

Endpoint análisis de ingredientes (OCR)

Método POST

Ruta: /analizar

Recibe una lista de ingredientes (ya procesados por OCR en la app) y los evalúa contra el perfil nutricional del usuario y la colección **ingredientes** de Firestore.

Registra el análisis tanto en una colección global como en un subdocumento del usuario.

¿Se necesita autenticación?

Si se necesita de un usuario identificado para usar el método.

Body (JSON):

```
{

“ingredientes”: [“harina de trigo”,” azúcar”,” leche en polvo”],

“producto”: “Galletas chocolate” // opcional

}
```

- Ingredientes: array obligatorio de string
- Producto: nombre del producto analizado (opcional, se usa para el historial)

Validaciones importantes:

- Si ingredientes no es array o está vacío -> 400 {“error”: “Faltan ingredientes en la solicitud”}
- Se carga el perfil del usuario desde usuarios/{uid} (rol, alergias, tipo_alimentacion, restricciones_personales)
- Se consulta la colección ingredientes para cada ingrediente normalizado

Respuesta 200 (ejemplo):

```
{
 "analisis": [
  {“ingrediente": "harina de trigo", "estado": "no_apto”},
  {“ingrediente": "azúcar”,       "estado": "apto"    },
  {“ingrediente": "leche en polvo”, “estado": "no_apto”}
 ]
}
```

Donde estado puede ser:

- “apto”
- “no apto”
- “desconocido” (si el ingrediente no existe en la colección)

Además, internamente se registra:

- En la colección global análisis_logs (solo visible para el admin)
- En usuarios/{uid}/historial_analisis

Endpoint Historial de usuario

Metodo: GET

Ruta: /usuario/historial

Devuelve los últimos análisis realizados por el usuario autenticado

¿Se necesita autenticación?

Si, se necesita de un usuario autenticado

Funcionamiento:

- Lee documentos desde usuarios/{uid}/análisis
- Ordena por createdAT descendente

Respuesta 200:


```
{
 "historial": [
  {
   "id": "abc123",
   "producto": "Galletas chocolate",
   "day": "2025-03-10",
   "resumen": {
    "total": 3,
    "no_apto": 2,
    "aptos": 1,
    "desconocido": 0
   },
   "createdAt": 1731270000000,
   "analisis": [
    {“ingrediente": "harina de trigo", "estado": "no_apto”},
    {“ingrediente": "azúcar”,       "estado": "apto"    },
    {“ingrediente": "leche en polvo”, “estado": "no_apto” }
   ],
   "foto_url": null
  }
 ]
}
```

Endpoints de administración

Todos estos endpoints:

- Usan verificar Token
- Llaman a verifiacarRol (uid," admin")
- Si el usuario no es admin -> 403 {"error": "Acceso denegado. No tienes permisos suficientes"}

Reporte global admin

Método GET

Ruta: /admin/reporte

Devuelve un resumen de los últimos análisis realizados por todos los usuarios

Respuesta 200 (ejemplo):


```
{
 "registros": [
  {
   "id": "log123",
   "day": "2025-03-10",
   "resumen": {"total": 3, "no_apto": 2, "aptos": 1, "desconocido": 0},
   "perfil": {"tipo_alimentacion": "Celiaca", "alergias": "Ninguna", ...},
   "createdAt": 1731270000000
  }
 ]
}
```


Crear ingrediente

Método: POST

Ruta: /admin/ingredientes

Crea un nuevo ingrediente en la colección ingredientes con sus banderas (apto/no apto para distintos perfiles) y se registra el cambio en ingredientes logs.

Body (JSON)

```
{
 "nombre": "Harina de arroz",
 "celiaco": true,
 "contiene_lactosa": false,
 "vegano": true,
 "vegetariano": true,
 "descripcion": "Harina apta para celíacos"
}
```

Reglas:

- Nombre: obligatorio
- El ID del documento se genera normalizando el nombre (minúsculas, sin tildes, sin espacios extra)
- Si ya existe un documento con ese ID -> 409 {“error”: “el ingrediente ya existe”}

Respuesta 201 (ejemplo):

```
{
 "mensaje": "Ingrediente creado correctamente.",
 "id": "harina_de_arroz",
 "data": {
  "celiaco": true,
  "contiene_lactosa": false,
  "vegano": true,
  "vegetariano": true,
```

```
  "descripcion": "Harina apta para celíacos"
 }
}
```

Actualizar ingrediente

Método: PUT

Ruta: admin/ingredientes/: nombre

Actualiza parcialmente las propiedades de un ingrediente existente.

Parámetro de ruta:

- :nombre -> Se normaliza internamente para construir el ID de Firestore

Body (JSON, todos opcionales)

```
{
 "celiaco": false,
 "contiene_lactosa": true,
 "vegano": false,
 "vegetariano": true,
 "descripcion": "Ahora contiene gluten y lactosa"
}
```

Respuestas importantes:

- Si el ingrediente no existe -> 404 {“error”: ingrediente no encontrado}
- Si la actualización es exitosa -> 200 {“mensaje”: “Ingrediente actualizado correctamente. “id”:” -----”,” updates”: {...}}

Además, registra el cambio en **ingredientes_logs** con before y after.

Eliminar ingredientes

Método: DELETE

Ruta: /admin/ingredientes/:nombre

Elimina el ingrediente indicado de la colección ingredientes y registra el evento en ingredientes_logs.

Respuestas:

- No encontrado -> 404 {“error”: “Ingrediente no encontrado”}
- Eliminado -> 200 {“mensaje”: “Ingrediente eliminado correctamente”, “id”: “...”}

Listar ingredientes

Método: GET

Ruta: /admin/ingredientes

Query params opcionales:

- Limit: número máximo de documentos a devolver

Ejemplo:

```
GET /admin/ingredientes?limit=50

Authorization: Bearer <ID_TOKEN_ADMIN>
```

Respuesta 200:

```
{
 "ingredientes": [
  {
   "id": "harina_de_trigo",
   "celiaco": false,
   "contiene_lactosa": false,
```

```
    "vegano": false,
    "vegetariano": true,
    "descripcion": ""
  }
 ]
}
```

6.Conclusion

El desarrollo del backend de la aplicación Kachate permitió construir una infraestructura sólida, escalable y segura para soportar las principales funcionalidades de la plataforma, especialmente el análisis automatizado de ingredientes y la gestión de la información nutricional de los usuarios.

Gracias al uso de Node.js con Express y la integración con los servicios de Firebase (Firestore, Authentication y Functions), se logro implementar una arquitectura eficiente basada en servicios, capaz de manejar múltiples solicitudes simultaneas, validar correctamente la identidad de los usuarios y limitar el acceso mediante un sistema de roles

El backend cumple un rol fundamental dentro de la aplicación, ya que es el encargado de:

- Procesar los resultados del OCR generado por la app móvil
- Evaluar la compatibilidad de los alimentos según el perfil de usuario.
- Gestionar los ingredientes clasificados desde el panel de administración.
- Registrar el historial de analisis y generar los reportes globales para el administrador.

Además, el uso de Firebase Emulators durante el desarrollo permitió probar y depurar el sistema sin afectar el entono productivo ni generar costos innecesarios, lo que facilito el flujo de trabajo más seguro y profesional.