SmartBite

Integrantes

- Nicole Ariadna Celemin Triana - @nicole1ariadna
- Yeiner Arwawingumu Zapata Vallejo - @ARWAWINGUMU
- Deiber David Gongora Hurtado - @DeiberD
- Camilo Andres Beltran Zabala - @cxbeltzz

ENTREGA_06 TESTING

En el presente trabajo se realizaron pruebas de integración y de comportamiento centradas en validar el flujo completo del sistema de recomendación de recetas. El enfoque principal fue el uso de pruebas automatizadas que verifican tanto el procesamiento previo de los datos (limpieza, filtrado y normalización) como la interacción entre los diferentes módulos responsables de generar recomendaciones coherentes. Para ello se utilizaron tests que comprueban la correcta filtración por ingredientes, la transformación y formateo de los datos recibidos del modelo, el funcionamiento del pipeline de recomendación cuando existen o no suficientes recetas, la integración del algoritmo de vecinos más cercanos y la consistencia del escalado nutricional utilizado para procesar cada receta. Asimismo, se implementaron pruebas para garantizar que la aplicación distribuya adecuadamente las recomendaciones por tipo de comida, eliminando duplicados y cumpliendo con el número esperado de recetas por categoría. En conjunto, estas pruebas permiten asegurar que todo el sistema opere de manera robusta en escenarios reales y con diferentes configuraciones de entrada.

TEST: test_model_extract_filter.py

Objetivo:

Asegurar que ‘extract_ingredient_filtered_data’ solo devuelva recetas que
contienen todos los ingredientes buscados sin importar mayúsculas/minúsculas.

Casos límite:
- Coincidencia exacta de todos los ingredientes -> válido
- Ingrediente faltante en la receta -> excluido
- Coincidencia con diferencias de capitalización -> válido

SmartBite

TEST : test_model_output_format.py

Objetivo:

Comprobar que ‘output_recommended_recipes’ transforma los campos en formato ‘c("...")’ a listas de Python y normaliza los valores faltantes en las Imágenes.

Casos límite:

- Ingredientes/instrucciones en formato R -> lista de strings
- Campo ‘RecipeImages’ como ‘None’ -> lista vacía
- Cadena con comillas escapadas -> se preservan en la salida

TEST : test_model_recommend_fallback.py

Objetivo:

Verificar que ‘recommend’ retorna ‘None’ cuando no hay suficientes recetas para satisfacer ‘n_neighbors’, evitando ejecutar el pipeline con datos insuficientes.

Casos límite:

- DataFrame con menos recetas que ‘n_neighbors’ -> ‘None’
- Ingredientes opcionales vacíos -> no afectan el resultado
- Vector de nutrientes válido con 9 elementos -> requisito para la llamada

TEST: test_model_recommend_integration.py

Objetivo:

Validar que ‘recommend’ genera vecinos cuando existen suficientes recetas y que el resultado mantiene la cantidad solicitada en ‘n_neighbors’.

Casos límite:

- DataFrame con más recetas que vecinos -> respuesta válida (DataFrame)
- Vector alineado con la primera receta -> dicha receta aparece en el top
- Parámetros personalizados (`n_neighbors`) -> controla tamaño del resultado

SmartBite

TEST: test_model_scaling_pipeline.py

Objetivo:

Comprobar que 'scaling' normaliza los atributos nutricionales y entrega un 'StandardScaler' utilizable posteriormente en el pipeline de recomendación.

Casos límite:

- Múltiples recetas con diferentes magnitudes -> medias aproximadas a 0
- Varianzas no nulas -> evita divisiones por cero en el escalador
- Número de columnas esperado (9 features nutricionales) -> mantiene forma

TEST:test_app_recommend_meals.py

Objetivo:

Garantizar que 'recommend_meals' construye la rejilla de comidas que usa la UI: debe solicitar datos al modelo, deduplicar recetas y asignar exactamente 'RECIPES_PER_MEAL' por cada comida solicitada.

Casos límite:

- 'meals_per_day=2' -> usa etiquetas 'breakfast' y 'lunch'}
- Suficientes recetas únicas -> todas las ranuras se llenan
- Duplicados iniciales -> se eliminan antes de repartir

ARCHIVO: test_app.py

TEST: test_register_success_message

Objetivo:

Verificar que la ruta /register procesa correctamente una solicitud de registro válida cuando el usuario completa todos los campos correctamente y acepta los términos, mostrando el mensaje de éxito esperado en la interfaz.

Casos límite:

- Envío de formulario completo -> válido: Todos los campos requeridos (name, email, password, confirm_password, terms) están presentes y son correctos.
- Redirección exitosa -> código 200: La respuesta final después de la redirección tiene un status code 200 (OK).
- Mensaje de confirmación en UI -> presente: El HTML de la respuesta contiene el texto exacto ¡Cuenta creada exitosamente para..., confirmando el éxito de la operación al usuario.

TEST: test_register_password_mismatch

Objetivo:

Garantizar que el sistema de validación de la ruta /register rechaza activamente un intento de registro si los campos password y confirm_password no coinciden, protegiendo al usuario de errores tipográficos.

Casos límite:

- Contraseñas no coincidentes -> inválido: El valor del campo password es diferente al de confirm_password.
- Recarga de formulario -> sin error de servidor: La aplicación no crashea, sino que vuelve a mostrar el formulario de registro con un código de estado 200.
- Mensaje de error específico -> presente: El HTML de la respuesta muestra el mensaje de error Las contraseñas no coinciden para informar al usuario sobre el problema exacto.

TEST: test_register_without_terms

Objetivo:

Asegurar que la lógica de negocio de la ruta /register obliga al usuario a aceptar los términos y condiciones, bloqueando el registro si el campo correspondiente no es enviado.

Casos límite:

- Campo terms ausente -> inválido: La solicitud POST se envía sin el campo terms, simulando que el checkbox no fue marcado.
- Persistencia en la página de registro -> sin redirección: El usuario no avanza en el flujo y permanece en la página de registro para corregir el error.

- Mensaje de error sobre términos -> presente: La interfaz de usuario muestra claramente el mensaje de error Debes aceptar los términos y condiciones, indicando la acción requerida.

ARCHIVO: utests_login.py

TEST: test_01_login_page_loads

Objetivo:

Verificar que la ruta /login se renderiza correctamente y está disponible para el usuario, mostrando el formulario de inicio de sesión sin errores de servidor ni problemas de conectividad.

Casos límite:

- Solicitud GET a /login → válido: La petición HTTP GET se ejecuta sin errores de conexión o problemas de routing.
- Código de estado 200 → esperado: El servidor responde exitosamente con status code 200 (OK), indicando que la página está disponible.
- Contenido "login" presente → verificado: El HTML de la respuesta contiene la palabra "login" en el cuerpo de la página, confirmando que se cargó la plantilla correcta.

TEST: test_02_login_valid_credentials_mock

Objetivo:

Garantizar que el flujo de autenticación de la ruta /login funciona correctamente cuando el usuario proporciona credenciales válidas (email y contraseña correctos), utilizando mocks para simular la respuesta de la base de datos sin dependencias externas.

Casos límite:

- Credenciales válidas simuladas → usuario autenticado: Se crea un mock de usuario con password=True indicando autenticación exitosa.

- Usuario mock completo → campos requeridos: El objeto simulado incluye todos los atributos necesarios (id, username, is_authenticated, is_active, is_anonymous, get_id).
- Redirección post-login → código 200: Después de un login exitoso, el usuario es redirigido a la página principal con status code 200.
- Sin mensajes de error → respuesta limpia: La respuesta no contiene mensajes de error como "Usuario Inválido" o "Contraseña Inválida".

TEST: test_03_login_invalid_password_mock

Objetivo:

Asegurar que el sistema de autenticación de la ruta /login rechaza correctamente intentos de inicio de sesión con contraseña incorrecta, mostrando un mensaje de error específico al usuario y evitando accesos no autorizados.

Casos límite:

- Contraseña incorrecta simulada → password = False: El mock devuelve un usuario con el campo password = False, simulando fallo en la verificación del hash.
- Permanencia en /login → sin redirección: El usuario permanece en la página de login para intentar nuevamente, respuesta con código 200.
- Mensaje de error específico → presente: El HTML contiene el texto "Contraseña" indicando explícitamente que la contraseña es inválida, no el usuario.