MY FOODIE

BRYAN ANDRÉS VILLABONA
SERGIO STEVEN LIÉVANO

GRUPO: U1

PROFESOR:
ING. PEDRO PEREZ
ING. JUAN MARIÑO

CAMPUSLANDS
SALON U1
RUTA NODE.JS
FLORIDABLANCA
2025

Tabla de contenido

My Foodie

1. Situación Problema

El ecosistema digital de reseñas gastronómicas, aunque saturado, presenta una barrera significativa para los comensales: **la falta de confianza y transparencia**. Los usuarios ("foodies") y el público en general se enfrentan a plataformas donde los rankings son a menudo opacos, fácilmente manipulables por reseñas falsas, o basados en promedios simples que no reflejan la calidad real o actual de un restaurante.

Conceptos abstractos como "ranking justo", "crítica relevante" o "calificación ponderada" son difíciles de percibir en sistemas donde una calificación de 5 estrellas de hace dos años tiene el mismo peso que una de 1 estrella de ayer. Esta dificultad genera desinterés, desconfianza en la comunidad y una toma de decisiones deficiente al elegir dónde comer.

El reto principal es, por tanto, traducir la "confianza" en un sistema tangible. Se necesita una plataforma que no solo recopile opiniones, sino que las pondere de forma inteligente e interactiva. "My Foodie" nace para solucionar este problema implementando un **sistema de ranking ponderado transparente**, donde la calificación de una reseña se ve influenciada por factores claros: la antigüedad (más nuevas valen más) y la validación de la comunidad (likes/dislikes), asegurando que las recomendaciones sean justas, relevantes y se mantengan actualizadas.

2. Levantamiento de Requerimientos

En el contexto de la formación Full-Stack, los desarrolladores en entrenamiento deben dominar no solo la creación de funcionalidades (CRUDs), sino también la implementación de arquitecturas robustas y desacopladas (Frontend/Backend), la gestión segura de la autenticación y la integridad de los datos en operaciones complejas.

Los requisitos de este taller exigen ir más allá de un simple "blog de recetas". Se exige una aplicación que maneje roles, aprobaciones, y un sistema de calificación complejo que debe ser transaccional para garantizar la consistencia de los datos.

Esta situación genera una necesidad clara: diseñar un producto digital que cumpla con especificaciones técnicas de alto nivel (Node.js con driver nativo de MongoDB, transacciones, JWT, JS Puro en el frontend), resolviendo al mismo tiempo un problema del mundo real (la confianza en las reseñas).

El proyecto "My Foodie" se diseñó para cumplir con estos desafíos, estableciendo los siguientes requerimientos:

2.1. Requerimientos

Para construir una solución efectiva, se realizó un levantamiento de requerimientos tomando como base:

1. **Las especificaciones del Taller (RNF):** ¿Qué tecnologías y arquitecturas son obligatorias? (driver nativo de MongoDB, API REST, JS Puro, Transacciones, etc.).
2. **La necesidad del Usuario Cliente (RF):** ¿Qué debe poder hacer el comensal? (Registrarse, ver restaurantes, filtrar, reseñar, votar).
3. **La necesidad del Administrador (RF):** ¿Qué debe poder hacer el gestor? (Aprobar restaurantes, gestionar categorías y platos).
4. **La lógica de Negocio (RF):** ¿Cómo funciona el sistema de Ranking Ponderado?

2.2. Requerimientos Funcionales

ID	Requerimiento Funcional	Descripción
RF01	Gestión de Autenticación	El sistema debe permitir a los usuarios registrarse (nombre, email, password) e iniciar sesión (email, password). El acceso a rutas protegidas se gestionará con JWT.
RF02	Gestión de Roles	El sistema debe manejar dos roles: "cliente" (rol por defecto) y "admin". Los permisos de API deben estar diferenciados.
RF03	CRUD de Categorías (Admin)	Un administrador debe poder Crear, Leer, Actualizar y Eliminar categorías (ej. "Comida Rápida", "Gourmet"). El sistema debe impedir eliminar una categoría si está en uso por un restaurante.
RF04	Flujo de Aprobación de Restaurantes	Un "cliente" solo puede *proponer* un restaurante (estado "pendiente"). Un "admin" debe poder ver la lista de restaurantes pendientes, "aprobarlos" (hacerlos públicos) o "rechazarlos" (eliminarlos).
RF05	CRUD de Restaurantes (Admin)	Un administrador debe poder Leer (todos), Actualizar (cualquiera) y Eliminar (cualquiera) restaurantes.

ID	Requerimiento Funcional	Descripción
RF06	CRUD de Platos (Admin)	Un administrador debe poder Crear, Leer, Actualizar y Eliminar platos asociados a un restaurante específico (aprobado).
RF07	Creación de Reseñas (Cliente)	Un "cliente" debe poder crear una reseña (calificación 1-5 y comentario) para un restaurante. El sistema debe impedir que un usuario cree más de una reseña por restaurante.
RF08	Gestión de Reseñas (Cliente)	Un "cliente" debe poder Actualizar o Eliminar *únicamente* sus propias reseñas.
RF09	Votación de Reseñas (Cliente)	Un "cliente" debe poder dar "like" o "dislike" a las reseñas de otros usuarios. El sistema debe impedir el autovoto (votar en sus propias reseñas).
RF10	Ranking Ponderado (Automático)	El sistema debe recalcular el rankingPonderado de un restaurante automáticamente (usando el script ranking.js) cada vez que se cree, actualice, elimine una reseña, o se registre un voto (like/dislike).
RF11	Listado y Filtro de Restaurantes	El sistema debe permitir a cualquier usuario ver los restaurantes "aprobados", filtrarlos por categoría y ordenarlos por "ranking", "popularidad" (total de reseñas) o "recientes".
RF12	Panel de Usuario (Cliente)	Un "cliente" autenticado debe poder ver su "Dashboard" con sus estadísticas (total reseñas, promedio) y una página con la lista de "Mis Reseñas".

2.3. Requerimientos No Funcionales

ID	Requerimiento No Funcional	Descripción
RNF01	Stack de Backend	La API debe estar desarrollada 100% en Node.js con Express.
RNF02	Stack de Frontend	El cliente web debe estar desarrollado 100% en HTML, CSS y JavaScript Puro (Vanilla JS).

ID	Requerimiento No Funcional	Descripción
RNF03	Base de Datos	La persistencia de datos debe ser en MongoDB, utilizando el driver oficial (mongodb).
RNF04	Integridad de Datos	Las operaciones críticas que afectan el ranking (gestión de reseñas y votos) deben estar envueltas en Transacciones de MongoDB (session.withTransaction()) para garantizar consistencia.
RNF05	Seguridad (Autenticación)	El sistema debe implementar autenticación basada en JWT, utilizando passport-jwt, jsonwebtoken y bcrypt para el hash de contraseñas.
RNF06	Seguridad (API)	La API debe estar protegida contra abuso de peticiones usando express-rate-limit.
RNF07	Validación de Datos (API)	Todos los endpoints que reciben datos (body, params) deben ser validados usando express-validator (DTOs).
RNF08	Documentación de API	Todos los endpoints del backend deben estar documentados y ser explorables usando swagger-ui-express.
RNF09	Configuración	El proyecto debe utilizar dotenv para la gestión de variables de entorno (Puerto, URI de Mongo, JWT Secret) y proveer un .env.example.
RNF10	Arquitectura	El proyecto debe estar dividido en dos repositorios independientes (Backend y Frontend). El backend debe tener una arquitectura modular (controllers, services, routes, dtos, etc.).
RNF11	Versionado	La API debe ser versionada (ej. /api/v1) y el package.json debe usar semver.
RNF12	Usabilidad (Frontend)	La interfaz de usuario debe ser intuitiva, amigable y responsive (adaptable a móviles).

2.4. Historias De Usuario Con Criterios De Aceptación

A continuación, se detallan las Historias de Usuario (HU) que definen el Product Backlog del proyecto "My Foodie", describiendo las funcionalidades desde la perspectiva del usuario (Rol) y estableciendo los Criterios de Aceptación para su validación.

HISTORIA DE USUARIO HU-01			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	**RF01**	**ACTOR**	**Usuario (No autenticado)**
NOMBRE DEL REQUERIMIENTO:	Registro de Nuevo Usuario		
DESCRIPCIÓN:			
Como usuario nuevo, quiero poder registrarme en la plataforma proporcionando mi nombre, email y una contraseña, para poder crear una cuenta y acceder a las funciones de cliente.			
FUNCIONALIDAD:			
Proveer un formulario público que permita a un visitante crear una nueva cuenta de usuario. El sistema debe validar los datos de entrada y almacenar al nuevo usuario con un rol de "cliente" por defecto.			
CRITERIOS DE ACEPTACIÓN:	**1. El sistema debe solicitar un nombre, un correo electrónico y una contraseña.** **2. El sistema debe validar que el correo electrónico no esté previamente registrado en la base de datos.** **3. El sistema debe validar que la contraseña cumpla con una longitud mínima (ej. 6 caracteres).** **4. El sistema debe almacenar la contraseña del usuario de forma segura (encriptada).** **5. Al completarse el registro exitosamente, el sistema debe informar al usuario y dirigirlo a la vista de inicio de sesión.**		
RESTRICCIONES:			
El correo electrónico debe ser único por cada usuario en la plataforma.			

HISTORIA DE USUARIO HU-02			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	**RF01**	**ACTOR**	**Usuario (registrado)**
NOMBRE DEL REQUERIMIENTO:	Inicio de Sesión de Usuario		
DESCRIPCIÓN:			

Como usuario registrado, quiero poder iniciar sesión usando mi email y contraseña, para que el sistema me reconozca y me dé acceso a las funcionalidades protegidas de la plataforma.
FUNCIONALIDAD:
Proveer un formulario que permita a un usuario registrado autenticarse mediante su email y contraseña. El sistema debe verificar las credenciales y, si son correctas, generar una sesión segura (token) para el usuario que le permita interactuar con la aplicación.

CRITERIOS DE ACEPTACIÓN:	1. El sistema debe validar que el correo electrónico exista en la base de datos de usuarios. 2. El sistema debe validar que la contraseña proporcionada sea correcta para ese correo electrónico. 3. Si las credenciales son incorrectas, el sistema debe mostrar un mensaje de error claro al usuario. 4. Si las credenciales son correctas, el sistema debe otorgar acceso y redirigir al usuario al panel principal (Dashboard) de la aplicación.

RESTRICCIONES:
El acceso se basa en la combinación exacta de email y contraseña. El sistema debe tener un mecanismo para prevenir ataques de fuerza bruta (límite de intentos).

HISTORIA DE USUARIO HU-03			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF01**	**ACTOR**	**Usuario (autenticado)**
NOMBRE DEL REQUERIMIENTO:	Cierre de Sesión (Logout)		
DESCRIPCIÓN:			
Como usuario autenticado, quiero poder cerrar mi sesión activa en cualquier momento, para proteger mi cuenta en un dispositivo compartido o al finalizar mi visita.			
FUNCIONALIDAD:			
Permitir al usuario finalizar su sesión activa mediante una opción (botón) en el menú de perfil. El sistema debe invalidar la sesión del usuario en el cliente (navegador) y redirigirlo a la pantalla de inicio de sesión.			

CRITERIOS DE ACEPTACIÓN:	**1. El sistema debe mostrar una opción clara y accesible de "Cerrar Sesión" (ej. en el menú de perfil).** **2. El sistema debe solicitar confirmación al usuario antes de proceder con el cierre de sesión.** **3. Al confirmar, el sistema debe eliminar la información de la sesión almacenada en el navegador.** **4. El sistema debe redirigir automáticamente al usuario a la pantalla de inicio de sesión (pública).**

RESTRICCIONES:
Esta opción solo debe ser visible para usuarios que ya han iniciado sesión.

HISTORIA DE USUARIO HU-04			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RNF05, RNF10**	**ACTOR**	**Sistema**
NOMBRE DEL REQUERIMIENTO:	Protección de Vistas (Autenticación)		
DESCRIPCIÓN:			
Como Sistema, quiero proteger todas las vistas privadas de la aplicación, para asegurar que solo los usuarios autenticados puedan acceder a ellas.			
FUNCIONALIDAD:			
Implementar un mecanismo de seguridad que verifique la existencia de una sesión de usuario válida (autenticación) antes de permitir la visualización de cualquier página interna (ej. Dashboard, Detalle de Restaurante, Perfil, Vistas de Admin).			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe interceptar cualquier intento de acceso a una vista privada. 2. Si el usuario *no* está autenticado, el sistema debe redirigirlo automáticamente a la página de inicio de sesión. 3. Si el usuario *sí* está autenticado, el sistema debe permitirle el acceso y mostrar la vista solicitada. 4. Si un usuario autenticado intenta visitar la página de inicio de sesión, el sistema debe redirigirlo automáticamente a su panel principal (Dashboard).		
RESTRICCIONES:			
La verificación de la sesión debe realizarse en el lado del cliente (frontend) para la navegación y ser re-validada en el servidor (backend) al solicitar datos.			

HISTORIA DE USUARIO HU-05			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	**RF02, RNF05**	**ACTOR**	**Sistema**
NOMBRE DEL REQUERIMIENTO:	Protección de Funciones (Autorización de Admin)		
DESCRIPCIÓN:			
Como Sistema, quiero proteger todas las funciones y vistas administrativas, para asegurar que solo los usuarios con rol de "admin" puedan acceder a ellas.			
FUNCIONALIDAD:			
Implementar un mecanismo de seguridad (autorización) que verifique si el usuario autenticado posee el rol de "admin", antes de permitir el acceso a vistas o la ejecución de acciones administrativas (ej. gestionar categorías, aprobar restaurantes).			

CRITERIOS DE ACEPTACIÓN:	**1. El sistema debe verificar el rol del usuario autenticado en cada solicitud a una función administrativa.** **2. Si el usuario es "admin", el sistema debe permitir el acceso a la vista o la ejecución de la acción.** **3. Si el usuario *no* es "admin" (es "cliente"), el sistema debe denegar el acceso.** **4. Al denegar el acceso, el sistema (frontend) debe redirigir al usuario a una página no autorizada o a su Dashboard.** **5. Al denegar el acceso, el sistema (backend) debe responder con un error claro de "Acceso Prohibido".**
RESTRICCIONES:	
Esta verificación de rol solo se aplica a usuarios que ya han sido autenticados (HU-04).	

HISTORIA DE USUARIO HU-06			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF03**	**ACTOR**	**Administrador**
NOMBRE DEL REQUERIMIENTO:	Visualizar Lista de Categorías		
DESCRIPCIÓN:			
Como Administrador, quiero poder ver una lista de todas las categorías de restaurantes que existen en el sistema, para tener una visión general de la clasificación actual.			
FUNCIONALIDAD:			
Proveer una vista en el panel de administración que muestre, en formato de tabla o lista, todas las categorías registradas en la base de datos. La lista debe incluir el nombre y la descripción de cada categoría, así como las opciones de gestión (Editar, Eliminar).			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe permitir que el administrador acceda a una sección de "Gestión de Categorías". 2. El sistema debe mostrar una tabla con las columnas "Nombre", "Descripción" y "Acciones". 3. El sistema debe poblar esta tabla con todas las categorías existentes. 4. Si no existen categorías, el sistema debe mostrar un mensaje claro indicándolo (ej. "No hay categorías creadas.").		
RESTRICCIONES:			
Esta vista solo debe ser accesible para usuarios con el rol de "Administrador".			

HISTORIA DE USUARIO HU-07			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF03**	**ACTOR**	**Administrador**

NOMBRE DEL REQUERIMIENTO:	Crear Nueva Categoría
DESCRIPCIÓN:	
Como Administrador, quiero poder agregar una nueva categoría al sistema (ej. "Vegetariana", "Sushi"), para clasificar nuevos tipos de restaurantes.	
FUNCIONALIDAD:	
Proveer un formulario (accesible desde la vista de "Gestión de Categorías") que permita al administrador ingresar el nombre y una descripción opcional para una nueva categoría. El sistema debe validar los datos antes de guardarla.	
CRITERIOS DE ACEPTACIÓN:	**1. El sistema debe proveer un botón o enlace de "Crear Nueva" en la vista de categorías.** **2. Al activarlo, el sistema debe mostrar un formulario (modal o en página) solicitando "Nombre" (obligatorio) y "Descripción".** **3. El sistema debe validar que el "Nombre" de la categoría no esté duplicado en la base de datos.** **4. Si el nombre ya existe, el sistema debe mostrar un mensaje de error y no permitir la creación.** **5. Al guardar exitosamente, la nueva categoría debe aparecer en la lista (HU-06) sin necesidad de recargar la página.**
RESTRICCIONES:	
El "Nombre" de la categoría debe ser único.	

HISTORIA DE USUARIO HU-08			
Prioridad: Baja			
CÓDIGO DEL REQUERIMIENTO:	**RF03**	**ACTOR**	**Administrador**
NOMBRE DEL REQUERIMIENTO:	Actualizar Categoría Existente		
DESCRIPCIÓN:			
Como Administrador, quiero poder editar el nombre o la descripción de una categoría que ya existe, para corregir errores o actualizar su definición.			
FUNCIONALIDAD:			
Permitir al administrador modificar los datos de una categoría específica. El sistema debe proveer un botón "Editar" en la lista de categorías (HU-06) que abra el formulario de edición (similar a HU-07) con la información actual de la categoría ya cargada.			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar un botón "Editar" para cada categoría en la lista. 2. Al hacer clic en "Editar", el sistema debe mostrar el formulario de edición con los campos "Nombre" y "Descripción" poblados con los datos actuales. 3. El administrador debe poder modificar el nombre y/o la descripción.		

	4. El sistema debe validar que el *nuevo* nombre (si se cambia) no esté duplicado en la base de datos. 5. Al guardar los cambios, la lista de categorías (HU-06) debe reflejar la información actualizada.
RESTRICCIONES:	
El identificador único (ID) de la categoría no puede ser modificado.	

HISTORIA DE USUARIO HU-09			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF03**	**ACTOR**	**Administrador**
NOMBRE DEL REQUERIMIENTO:	Eliminar Categoría (con validación)		
DESCRIPCIÓN:			
Como Administrador, quiero poder eliminar una categoría que ya no es necesaria o fue creada por error, para mantener limpia la taxonomía del sistema.			
FUNCIONALIDAD:			
Proveer un botón "Eliminar" en la lista de categorías (HU-06). Antes de borrar, el sistema debe realizar una validación de negocio clave: verificar que la categoría no esté siendo utilizada por ningún restaurante.			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar un botón "Eliminar" para cada categoría en la lista. 2. Al hacer clic en "Eliminar", el sistema debe solicitar una confirmación al administrador (ej. "¿Estás seguro?"). 3. Al confirmar, el sistema debe verificar internamente si algún restaurante en la base de datos está asociado a esta categoría. 4. Si la categoría *está en uso*, el sistema debe mostrar un mensaje de error (ej. "No se puede eliminar, categoría en uso") y cancelar la eliminación. 5. Si la categoría *no está en uso*, el sistema debe eliminarla permanentemente de la base de datos.		
RESTRICCIONES:			
El sistema debe priorizar la integridad de los datos; solo se pueden eliminar categorías que no tengan restaurantes asociados.			

HISTORIA DE USUARIO HU-10			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF04**	**ACTOR**	**Cliente - Administrador**
NOMBRE DEL REQUERIMIENTO:	Proponer Nuevo Restaurante		

DESCRIPCIÓN:	
Como Cliente, quiero poder proponer un restaurante que no se encuentra en la plataforma, para que los administradores lo revisen y, si es aprobado, sea visible para todos.	
FUNCIONALIDAD:	
Proveer un formulario (accesible desde un botón "Proponer Restaurante" en el layout) donde el cliente pueda ingresar los detalles de un restaurante (nombre, descripción, ubicación, categoría e imagen opcional). La propuesta se guarda en el sistema con un estado "pendiente".	
CRITERIOS DE ACEPTACIÓN:	**1. El sistema debe mostrar un botón "Proponer Restaurante" a todos los usuarios autenticados.** **2. El formulario de propuesta debe solicitar: Nombre (obligatorio), Descripción (obligatorio), Ubicación (obligatorio) y Categoría (obligatorio).** **3. El campo "Categoría" debe ser un selector (<select>) que muestre la lista de categorías existentes (obtenidas de HU-06).** **4. El sistema debe validar que el "Nombre" del restaurante no esté duplicado en la base de datos.** **5. Al enviar, el sistema debe guardar el restaurante con un estado "pendiente" (no visible en la lista pública).** **6. El sistema debe mostrar una notificación al usuario confirmando que "Su propuesta ha sido enviada a revisión".**
RESTRICCIONES:	
Esta acción solo crea la propuesta; no publica el restaurante.	

HISTORIA DE USUARIO HU-11			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF04**	**ACTOR**	**Administrador**
NOMBRE DEL REQUERIMIENTO:	Visualizar Lista de Restaurantes Pendientes		
DESCRIPCIÓN:			
Como Administrador, quiero poder ver una lista de todos los restaurantes que los clientes han propuesto y que están "pendientes" de revisión, para gestionar la cola de aprobación.			
FUNCIONALIDAD:			
Proveer una vista en el panel de administración ("Aprobaciones") que muestre una tarjeta informativa por cada restaurante que actualmente tenga el estado "pendiente".			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe permitir que el administrador acceda a una sección de "Aprobaciones". 2. El sistema debe consultar y mostrar *únicamente* los restaurantes cuyo estado sea "pendiente". 3. Cada restaurante en la lista debe mostrar su nombre, descripción, ubicación, categoría e imagen propuesta.		

	4. Cada restaurante en la lista debe tener opciones visibles para "Aprobar" y "Rechazar". 5. Si no hay restaurantes pendientes, el sistema debe mostrar un mensaje claro (ej. "¡Todo al día! No hay restaurantes pendientes.").
RESTRICCIONES:	
Esta lista solo debe ser accesible para usuarios "admin" y no debe mostrar restaurantes ya "aprobados".	

HISTORIA DE USUARIO HU-12			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF04**	**ACTOR**	**Administrador**
NOMBRE DEL REQUERIMIENTO:	Aprobar Restaurante Pendiente		
DESCRIPCIÓN:			
Como Administrador, quiero poder "Aprobar" un restaurante de la lista de pendientes, para que se vuelva público y sea visible para todos los usuarios en la plataforma.			
FUNCIONALIDAD:			
Proveer un botón "Aprobar" en la tarjeta de un restaurante pendiente (HU-11). Al utilizarlo, el sistema debe actualizar el estado del restaurante de "pendiente" a "aprobado".			
CRITERIOS DE ACEPTACIÓN:	**1. El sistema debe permitir al administrador hacer clic en el botón "Aprobar" de un restaurante pendiente.** **2. Al confirmar la acción, el estado del restaurante en la base de datos debe cambiar de "pendiente" a "aprobado".** **3. Una vez aprobado, el restaurante debe desaparecer inmediatamente de la lista de "Aprobaciones pendientes".** **4. El restaurante ahora "aprobado" debe ser visible en la lista pública de restaurantes (HU-16).**		
RESTRICCIONES:			
Esta acción solo modifica el estado del restaurante.			

HISTORIA DE USUARIO HU-13			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF04**	**ACTOR**	**Administrador**
NOMBRE DEL REQUERIMIENTO:	Rechazar Restaurante Pendiente		
DESCRIPCIÓN:			

Como Administrador, quiero poder "Rechazar" una propuesta de restaurante (ej. es spam, duplicado, inapropiado), para eliminarla permanentemente del sistema.	
FUNCIONALIDAD:	
Proveer un botón "Rechazar" en la tarjeta de un restaurante pendiente (HU-11). Al confirmarlo, el sistema debe eliminar el registro del restaurante de la base de datos.	
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe permitir al administrador hacer clic en el botón "Rechazar" de un restaurante pendiente. 2. El sistema debe solicitar una confirmación al administrador antes de proceder con la eliminación. 3. Al confirmar, el restaurante (con estado "pendiente") debe ser eliminado permanentemente de la base de datos. 4. La tarjeta del restaurante debe desaparecer de la lista de "Aprobaciones pendientes".
RESTRICCIONES:	
Esta acción es destructiva y no se puede deshacer.	

HISTORIA DE USUARIO HU-14			
Prioridad: Baja			
CÓDIGO DEL REQUERIMIENTO:	**RF05**	**ACTOR**	**Administrador**
NOMBRE DEL REQUERIMIENTO:	Actualizar Restaurante Aprobado		
DESCRIPCIÓN:			
Como Administrador, quiero poder editar la información (nombre, descripción, categoría, etc.) de un restaurante que ya está público, para corregir errores o actualizar sus datos.			
FUNCIONALIDAD:			
Proveer una opción "Editar Restaurante" en la página de detalle de un restaurante (solo visible para admins). Esta acción debe abrir un formulario con la información actual del restaurante, permitiendo al administrador modificarla y guardarla.			
CRITERIOS DE ACEPTACIÓN:	**1. El sistema debe mostrar un botón "Editar Restaurante" en la página de detalle del restaurante *solo* a los usuarios con rol "admin".** **2. Al activar esta opción, el sistema debe mostrar un formulario de edición precargado con los datos existentes del restaurante (nombre, descripción, ubicación, categoría, imagen).** **3. El sistema debe permitir al administrador modificar cualquiera de estos campos.** **4. El sistema debe validar que el nuevo nombre (si se cambia) no esté duplicado en la base de datos.** **5. Al guardar, los cambios deben persistir y ser visibles en la página de detalle del restaurante.**		

RESTRICCIONES:	
Esta función es para editar restaurantes que ya son públicos (estado "aprobado").	

HISTORIA DE USUARIO HU-15			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	**RF05**	**ACTOR**	**Administrador**
NOMBRE DEL REQUERIMIENTO:	Eliminar Restaurante Aprobado (y sus datos asociados)		
DESCRIPCIÓN:			
Como Administrador, quiero poder eliminar permanentemente un restaurante que ya está aprobado (ej. el negocio cerró), asegurándome de que todas sus reseñas y platos también se eliminen.			
FUNCIONALIDAD:			
Proveer una opción "Eliminar Restaurante" en la página de detalle (solo para admins). Esta acción debe ejecutar una eliminación en cascada, borrando el restaurante y toda la información vinculada a él (platos y reseñas) de forma transaccional.			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar un botón "Eliminar Restaurante" en la página de detalle *solo* a los usuarios con rol "admin". 2. El sistema debe mostrar una advertencia clara de que esta acción es destructiva y eliminará *todos* los datos asociados (platos, reseñas). 3. Al confirmar, el sistema debe eliminar el restaurante. 4. El sistema debe garantizar (mediante una operación transaccional) que todos los registros de platos asociados a ese restaurante sean eliminados. 5. El sistema debe garantizar (mediante la misma operación transaccional) que todos los registros de reseñas asociados a ese restaurante sean eliminados.		
RESTRICCIONES:			
Esta operación es la más destructiva del sistema y debe ser transaccional para prevenir datos huérfanos en la base de datos.			

HISTORIA DE USUARIO HU-16			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	**RF11**	**ACTOR**	**Usuario (Autenticado)**
NOMBRE DEL REQUERIMIENTO:	Visualizar Lista Pública de Restaurantes		
DESCRIPCIÓN:			
Como usuario, quiero ver una lista de todos los restaurantes que han sido aprobados, para poder explorarlos y decidir cuál me interesa visitar.			

FUNCIONALIDAD:	
Proveer una vista principal de "Explorar" que consulte la base de datos y muestre una cuadrícula (grid) de tarjetas, una por cada restaurante. Esta vista es la principal para el descubrimiento de lugares.	
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe permitir que el usuario acceda a una sección de "Explorar" desde la navegación principal. 2. El sistema debe consultar y mostrar *únicamente* los restaurantes cuyo estado sea "aprobado". 3. Cada restaurante debe mostrarse como una "tarjeta" visual que incluya su imagen, nombre y la categoría a la que pertenece. 4. El sistema debe mostrar la calificación de ranking (ponderada) de cada restaurante en su tarjeta. 5. Si hay muchos restaurantes, el sistema debe gestionar la visualización mediante paginación.
RESTRICCIONES:	
Los restaurantes que estén en estado "pendiente" (HU-10) no deben ser visibles en esta lista pública.	

HISTORIA DE USUARIO HU-17			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	**RF11**	**ACTOR**	**Usuario (Autenticado)**
NOMBRE DEL REQUERIMIENTO:	Filtrar Restaurantes por Categoría		
DESCRIPCIÓN:			
Como usuario, quiero poder filtrar la lista de restaurantes por una categoría específica (ej. "Comida Rápida"), para encontrar solo los lugares que coinciden con mi interés.			
FUNCIONALIDAD:			
El sistema debe mostrar una lista de filtros (basados en las categorías existentes, HU-06) en la vista "Explorar". Al seleccionar un filtro, la lista de restaurantes (HU-16) debe actualizarse para mostrar solo los que pertenecen a esa categoría.			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar una lista de botones o enlaces de filtro, incluyendo una opción para "Todos" y una opción por cada categoría registrada (HU-06). 2. Por defecto, el filtro "Todos" debe estar seleccionado al cargar la vista. 3. Al hacer clic en un filtro de categoría, el sistema debe actualizar la lista de restaurantes (HU-16) mostrando solo los que pertenecen a la categoría seleccionada. 4. El sistema debe resaltar visualmente el filtro de categoría que está actualmente activo.		
RESTRICCIONES:			

Los filtros de categoría deben cargarse dinámicamente; si se crea una nueva categoría (HU-07), debe aparecer como filtro.

HISTORIA DE USUARIO HU-18			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF11**	**ACTOR**	**Usuario (Autenticado)**
NOMBRE DEL REQUERIMIENTO:	Ordenar Lista de Restaurantes		
DESCRIPCIÓN:			
Como usuario, quiero poder ordenar la lista de restaurantes por diferentes criterios (ej. mejor ranking, más popular, más reciente), para ver los más relevantes según mi preferencia.			
FUNCIONALIDAD:			
El sistema debe proveer un selector (dropdown) en la vista "Explorar" que permita al usuario cambiar el criterio de ordenamiento de la lista de restaurantes (HU-16).			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar un selector de ordenamiento con las opciones: "Ranking", "Popularidad" y "Recientes". 2. Por defecto, la lista debe estar ordenada por "Ranking" (del más alto al más bajo). 3. Al seleccionar "Popularidad", el sistema debe reordenar la lista mostrando primero los restaurantes que tengan un mayor número de reseñas. 4. Al seleccionar "Recientes", el sistema debe reordenar la lista mostrando primero los restaurantes que fueron "aprobados" más recientemente.		
RESTRICCIONES:			
El criterio de ordenamiento seleccionado debe aplicarse en conjunto con el filtro de categoría (HU-17) que esté activo.			

HISTORIA DE USUARIO HU-19			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF11**	**ACTOR**	**Usuario (Autenticado)**
NOMBRE DEL REQUERIMIENTO:	Buscar Restaurantes por Nombre		
DESCRIPCIÓN:			
Como usuario, quiero poder escribir el nombre (o parte del nombre) de un restaurante en una barra de búsqueda, para encontrarlo rápidamente en la lista.			
FUNCIONALIDAD:			

El sistema debe proveer un campo de texto (barra de búsqueda) en la vista "Explorar". A medida que el usuario escribe, la lista de restaurantes visible debe filtrarse para mostrar solo los que coinciden con el término de búsqueda.	
CRITERIOS DE ACE PTACIÓN:	1. El sistema debe mostrar una barra de búsqueda prominente en la vista "Explorar". 2. A medida que el usuario escribe en la barra (ej. "piz"), la lista de restaurantes (HU-16) debe actualizarse en tiempo real para mostrar solo los que contienen "piz" en su nombre. 3. La búsqueda debe ser insensible a mayúsculas y minúsculas (ej. "CORRAL" debe encontrar "El Corral"). 4. Si ningún restaurante coincide con la búsqueda, el sistema debe mostrar un mensaje claro (ej. "No se encontraron restaurantes que coincidan con tu búsqueda.").
RESTRICCIONES:	
Esta búsqueda debe operar sobre los resultados ya filtrados por categoría (HU-17) y ordenados (HU-18).	

HISTORIA DE USUARIO HU-20			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	**RF11**	**ACTOR**	**Usuario (Autenticado)**
NOMBRE DEL REQUERIMIENTO:	Ver Página de Detalle de un Restaurante		
DESCRIPCIÓN:			
Como usuario, quiero poder hacer clic en la tarjeta de un restaurante de la lista, para navegar a una página de "Detalle" donde pueda ver toda su información ampliada, sus platos y sus reseñas.			
FUNCIONALIDAD:			
Permitir la navegación desde la vista "Explorar" (HU-16) a una vista de detalle individual para cada restaurante. Esta vista debe cargar y mostrar toda la información pública y asociada del restaurante seleccionado.			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe hacer que cada tarjeta de restaurante en la lista (HU-16) sea un enlace clicable. 2. Al hacer clic, el sistema debe llevar al usuario a una nueva página (vista de "Detalle") específica para ese restaurante. 3. Esta vista debe cargar y mostrar la información completa del restaurante (imagen grande, nombre, descripción detallada, ubicación, categoría). 4. Esta vista debe incluir una sección dedicada a listar los platos (HU-21) de ese restaurante. 5. Esta vista debe incluir una sección dedicada a listar las reseñas (HU-25) de ese restaurante.		
RESTRICCIONES:			

El usuario solo puede ver el detalle de restaurantes que estén "aprobados".

HISTORIA DE USUARIO HU-21			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	**RF06**	**ACTOR**	**Usuario (Cliente y Admin)**
NOMBRE DEL REQUERIMIENTO:	Visualizar Platos de un Restaurante		
DESCRIPCIÓN:			
Como usuario, al visitar la página de detalle de un restaurante, quiero ver una lista de los platos que ofrece, para conocer su menú y precios.			
FUNCIONALIDAD:			
El sistema debe cargar y mostrar la lista de platos (nombre, descripción, precio, imagen) asociados al restaurante que se está visitando en la página de detalle (HU-20).			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar una sección "Platos" en la página de detalle del restaurante. 2. El sistema debe consultar y listar todos los platos asociados a ese restaurante en específico. 3. Cada plato en la lista debe mostrar su imagen (o una por defecto), nombre, descripción y precio. 4. Si el restaurante aún no tiene platos registrados, el sistema debe mostrar un mensaje claro (ej. "Aún no hay platos registrados.").		
RESTRICCIONES:			
Esta funcionalidad es de solo lectura para los usuarios con rol "Cliente".			

HISTORIA DE USUARIO HU-22			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF06**	**ACTOR**	**Administrador**
NOMBRE DEL REQUERIMIENTO:	Añadir Nuevo Plato a un Restaurante		
DESCRIPCIÓN:			
Como Administrador, al estar en la página de detalle de un restaurante, quiero poder añadir un nuevo plato a su menú, para expandir la oferta del restaurante.			
FUNCIONALIDAD:			
Proveer un botón "Añadir Plato" (visible solo para admins) en la página de detalle (HU-20). Este botón debe abrir un formulario (modal) que permita ingresar el nombre, descripción, precio y una URL de imagen opcional para un nuevo plato.			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar un botón "Añadir Plato" en la lista de platos (HU-21) *solo* si el usuario tiene el rol "Admin".		

	2. El formulario de creación debe solicitar Nombre (obligatorio), Descripción (obligatoria) y Precio (obligatorio, numérico positivo). 3. El sistema debe validar que el "Nombre" del plato no esté duplicado *dentro del mismo restaurante*. 4. Al guardar exitosamente, el nuevo plato debe aparecer en la lista de platos (HU-21) de la interfaz sin necesidad de recargar la página.
RESTRICCIONES:	
El sistema solo debe permitir añadir platos a restaurantes que ya tengan el estado "Aprobado".	

HISTORIA DE USUARIO HU-23			
Prioridad: Baja			
CÓDIGO DEL REQUERIMIENTO:	**RF06**	**ACTOR**	**Administrador**
NOMBRE DEL REQUERIMIENTO:	Editar Plato Existente		
DESCRIPCIÓN:			
Como Administrador, quiero poder editar la información de un plato existente (ej. su precio, descripción), para corregir errores o actualizar su definición.			
FUNCIONALIDAD:			
Proveer un botón "Editar" en cada plato de la lista (HU-21), visible solo para administradores. Este botón debe abrir el formulario de edición (similar a HU-22) con la información actual del plato ya cargada.			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar un botón "Editar" en cada plato de la lista *solo* para usuarios "admin". 2. Al hacer clic, el formulario de edición debe mostrar los datos actuales del plato (nombre, descripción, precio, imagen). 3. El sistema debe permitir al administrador modificar cualquiera de estos campos. 4. El sistema debe validar que el *nuevo* nombre (si se cambia) no esté duplicado *dentro del mismo restaurante*. 5. Al guardar los cambios, la lista de platos (HU-21) debe reflejar la información actualizada.		
RESTRICCIONES:			
El plato se actualiza sobre su propio identificador único (ID).			

HISTORIA DE USUARIO HU-24			
Prioridad: Baja			
CÓDIGO DEL REQUERIMIENTO:	**RF06**	**ACTOR**	**Administrador**
NOMBRE DEL REQUERIMIENTO:	Eliminar Plato Existente		

DESCRIPCIÓN:	
Como Administrador, quiero poder eliminar un plato de un restaurante, para quitarlo permanentemente del menú (ej. ya no se ofrece).	
FUNCIONALIDAD:	
Proveer un botón "Eliminar" en cada plato de la lista (HU-21), visible solo para administradores. El sistema debe solicitar confirmación antes de proceder con la eliminación permanente.	
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar un botón "Eliminar" en cada plato de la lista *solo* para usuarios "admin". 2. El sistema debe solicitar confirmación al administrador antes de borrar. 3. Al confirmar, el plato debe ser eliminado permanentemente de la base de datos. 4. El plato debe desaparecer de la lista (HU-21) en la interfaz del usuario.
RESTRICCIONES:	
Esta acción es destructiva e irreversible.	

HISTORIA DE USUARIO HU-25			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	**RF07, RF10**	**ACTOR**	**Cliente**
NOMBRE DEL REQUERIMIENTO:	Publicar Nueva Reseña		
DESCRIPCIÓN:			
Como Cliente, al ver el detalle de un restaurante, quiero poder publicar mi opinión (calificación de 1 a 5 estrellas y un comentario), para compartir mi experiencia e influir en el ranking.			
FUNCIONALIDAD:			
Proveer un formulario "Deja tu opinión" en la página de detalle del restaurante (HU-20). El sistema debe validar que el usuario no haya reseñado este lugar anteriormente. La creación de la reseña debe activar el recálculo del ranking del restaurante.			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar el formulario de reseña solo si el usuario autenticado *no* ha publicado una reseña para este restaurante específico. 2. Si el usuario ya publicó una reseña, el sistema debe ocultar el formulario y mostrar un mensaje (ej. "Ya has enviado una reseña."). 3. El formulario debe permitir al usuario seleccionar una calificación de 1 a 5 estrellas (obligatorio). 4. El formulario debe permitir al usuario escribir un comentario (obligatorio).		

	5. Al enviar la reseña, el sistema debe guardarla y asociarla al usuario y al restaurante. 6. El sistema debe garantizar (mediante una operación transaccional) que, inmediatamente después de guardar la reseña, el ranking ponderado del restaurante sea recalculado.
RESTRICCIONES:	
Un usuario solo puede crear una (1) reseña por restaurante.	

HISTORIA DE USUARIO HU-26			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	**RF11**	**ACTOR**	**Usuario (Autenticado)**
NOMBRE DEL REQUERIMIENTO:	Visualizar Lista de Reseñas de un Restaurante		
DESCRIPCIÓN:			
Como usuario, al visitar la página de detalle de un restaurante, quiero ver la lista de reseñas (comentarios y calificaciones) que otros usuarios han dejado, para tomar una decisión informada.			
FUNCIONALIDAD:			
El sistema debe cargar y mostrar la lista de reseñas asociadas al restaurante que se está visitando (HU-20). Debe mostrar la información del autor, su comentario, la calificación en estrellas, la fecha de publicación y los contadores de votos (likes/dislikes).			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar una sección "Reseñas" en la página de detalle del restaurante. 2. El sistema debe consultar y mostrar todas las reseñas asociadas a ese restaurante. 3. Cada reseña en la lista debe incluir el nombre del autor, la fecha, la calificación (de 1 a 5 estrellas) y el texto del comentario. 4. Cada reseña debe mostrar el número actual de "Likes" y "Dislikes". 5. Si el restaurante aún no tiene reseñas, el sistema debe mostrar un mensaje claro (ej. "Sé el primero en dejar una reseña.").		
RESTRICCIONES:			
Solo se muestran reseñas de restaurantes que están en estado "Aprobado".			

HISTORIA DE USUARIO HU-27			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF08, RF10**	**ACTOR**	**Cliente**
NOMBRE DEL REQUERIMIENTO:	Editar Reseña Propia		
DESCRIPCIÓN:			

Como Cliente, quiero poder editar el contenido (tanto la calificación como el comentario) de una reseña que yo publiqué previamente, para corregir un error o actualizar mi opinión.
FUNCIONALIDAD:
Permitir al cliente modificar sus propias reseñas. El sistema debe mostrar un botón "Editar" *solo* en las reseñas que pertenecen al usuario autenticado. Al activarlo, se debe presentar un formulario (modal) con la información actual de la reseña para su modificación.

CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar un botón "Editar" únicamente en las reseñas donde el autor coincide con el usuario autenticado. 2. Al hacer clic en "Editar", el sistema debe abrir un formulario de edición precargado con la calificación y el comentario actuales de esa reseña. 3. El sistema debe permitir al usuario cambiar la calificación (1-5) y/o el texto del comentario. 4. Al guardar los cambios, el sistema debe actualizar la reseña en la base de datos. 5. El sistema debe garantizar (mediante una operación transaccional) que, tras la actualización, el ranking ponderado del restaurante sea recalculado.

RESTRICCIONES:
El usuario no puede editar reseñas que pertenezcan a otras personas.

HISTORIA DE USUARIO HU-28			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF08, RF10**	**ACTOR**	**Cliente**
NOMBRE DEL REQUERIMIENTO:	Eliminar Reseña Propia		
DESCRIPCIÓN:			
Como Cliente, quiero poder eliminar permanentemente una reseña que yo publiqué (ej. si el restaurante ha cambiado mucho y mi opinión ya no es válida), para quitarla de la vista pública.			
FUNCIONALIDAD:			
Permitir al cliente eliminar sus propias reseñas. El sistema debe mostrar un botón "Eliminar" *solo* en las reseñas que pertenecen al usuario autenticado. El sistema debe solicitar confirmación antes de proceder con la eliminación.			

CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar un botón "Eliminar" únicamente en las reseñas donde el autor coincide con el usuario autenticado. 2. El sistema debe solicitar una confirmación clara al usuario antes de borrar la reseña. 3. Al confirmar, la reseña debe ser eliminada permanentemente de la base de datos.

	4. El sistema debe garantizar (mediante una operación transaccional) que, tras la eliminación, el ranking ponderado del restaurante sea recalculado. 5. Tras la eliminación, el usuario debe poder publicar una nueva reseña si lo desea (ver HU-25).
RESTRICCIONES:	
El usuario no puede eliminar reseñas de otras personas. Esta acción es irreversible.	

HISTORIA DE USUARIO HU-29			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF09, RF10**	**ACTOR**	**Cliente**
NOMBRE DEL REQUERIMIENTO:	Votar "Like" en Reseña Ajena		
DESCRIPCIÓN:			
Como Cliente, quiero poder dar "Like" (me gusta) a una reseña de otro usuario, para indicar que su opinión me parece útil o acertada e influir en la relevancia de esa reseña.			
FUNCIONALIDAD:			
Proveer un botón "Like" en todas las reseñas que no pertenecen al usuario autenticado. El sistema debe registrar el voto, actualizar el contador de likes y recalcular el ranking ponderado del restaurante, que se ve afectado por los votos.			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar un botón "Like" (con su contador) en las reseñas de otros usuarios. 2. El botón "Like" debe estar visible pero *deshabilitado* en las reseñas propias del usuario. 3. Al hacer clic en "Like", el sistema debe registrar el voto. Si el usuario ya había dado "Dislike" (HU-30) a esa reseña, ese voto debe retirarse. 4. El sistema debe actualizar el estado visual del botón (ej. "seleccionado") y el contador de "Likes". 5. El sistema debe garantizar (mediante una operación transaccional) que, tras el voto, el ranking ponderado del restaurante sea recalculado.		
RESTRICCIONES:			
El usuario no puede dar "Like" a sus propias reseñas.			

HISTORIA DE USUARIO HU-30			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF00, RF10**	**ACTOR**	**Cliente**

NOMBRE DEL REQUERIMIENTO:	Votar "Dislike" en Reseña Ajena
DESCRIPCIÓN:	
Como Cliente, quiero poder dar "Dislike" (no me gusta) a una reseña de otro usuario, para indicar que su opinión no me parece útil o es desacertada e influir en la relevancia de esa reseña.	
FUNCIONALIDAD:	
Proveer un botón "Dislike" en todas las reseñas que no pertenecen al usuario autenticado. El sistema debe registrar el voto, actualizar el contador de dislikes y recalcular el ranking ponderado del restaurante.	
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar un botón "Dislike" (con su contador) en las reseñas de otros usuarios. 2. El botón "Dislike" debe estar visible pero *deshabilitado* en las reseñas propias del usuario. 3. Al hacer clic en "Dislike", el sistema debe registrar el voto. Si el usuario ya había dado "Like" (HU-29) a esa reseña, ese voto debe retirarse. 4. El sistema debe actualizar el estado visual del botón (ej. "seleccionado") y el contador de "Dislikes". 5. El sistema debe garantizar (mediante una operación transaccional) que, tras el voto, el ranking ponderado del restaurante sea recalculado.
RESTRICCIONES:	
El usuario no puede dar "Dislike" a sus propias reseñas.	

HISTORIA DE USUARIO HU-31			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF12**	**ACTOR**	**Cliente**
NOMBRE DEL REQUERIMIENTO:	Visualizar Estadísticas del Dashboard		
DESCRIPCIÓN:			
Como Cliente, al iniciar sesión, quiero ser llevado a un "Dashboard" o panel principal donde pueda ver un resumen de mis estadísticas de actividad, para tener una vista rápida de mi participación.			
FUNCIONALIDAD:			
Proveer una vista de "Dashboard" como página principal para el usuario autenticado. Esta vista debe consultar y mostrar en tarjetas separadas las estadísticas clave del usuario: el número total de reseñas que ha escrito, su calificación promedio y el nombre de su restaurante favorito.			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe redirigir al usuario al "Dashboard" inmediatamente después de un inicio de sesión exitoso. 2. El sistema debe consultar y mostrar el recuento total de reseñas publicadas por el usuario.		

	3. El sistema debe calcular y mostrar la calificación promedio, basada en todas las reseñas del usuario (ej. "4.2"). 4. El sistema debe identificar y mostrar el nombre del restaurante al que el usuario otorgó su calificación más alta (considerado "Tu Favorito"). 5. Si el usuario no tiene reseñas, el sistema debe mostrar valores por defecto (ej. 0 reseñas, promedio 0, favorito "N/A").
RESTRICCIONES:	
Las estadísticas mostradas deben pertenecer únicamente al usuario que está autenticado.	

HISTORIA DE USUARIO HU-32			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RF12**	**ACTOR**	**Cliente**
NOMBRE DEL REQUERIMIENTO:	Visualizar mi Lista Personal de Reseñas		
DESCRIPCIÓN:			
Como Cliente, quiero poder acceder a una página de "Mis Reseñas", para ver un historial de todas las opiniones que he publicado, junto con el restaurante al que pertenecen.			
FUNCIONALIDAD:			
Proveer un enlace en la navegación principal a una vista de "Mis Reseñas". Esta página debe consultar y listar todas las reseñas creadas por el usuario autenticado, mostrando la información de la reseña y la información básica del restaurante asociado.			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar un enlace de navegación "Mis Reseñas" en el menú del usuario. 2. Al acceder a esta vista, el sistema debe mostrar una lista de todas las reseñas publicadas por el usuario, ordenadas por fecha (de la más reciente a la más antigua). 3. Cada elemento de la lista debe mostrar el nombre del restaurante, la calificación (estrellas) que el usuario otorgó, el comentario y la fecha. 4. Cada elemento debe incluir un enlace o botón para navegar directamente a la página de "Detalle" (HU-20) del restaurante correspondiente. 5. Si el usuario no ha publicado ninguna reseña, el sistema debe mostrar un mensaje claro invitándolo a explorar.		
RESTRICCIONES:			
Esta vista es personal y solo debe mostrar las reseñas del usuario autenticado.			

HISTORIA DE USUARIO HU-33
Prioridad: Alta

CÓDIGO DEL REQUERIMIENTO:	RNF06	ACTOR	Sistema
NOMBRE DEL REQUERIMIENTO:	Protección contra Abuso de API (Límite de Peticiones)		
DESCRIPCIÓN:			
Como Sistema, quiero limitar la cantidad de solicitudes (peticiones API) que una misma dirección IP puede realizar en un período de tiempo corto, para proteger los recursos del servidor y prevenir ataques.			
FUNCIONALIDAD:			
Implementar un mecanismo de "rate limiting" a nivel de servidor que rastree el número de peticiones por IP. Si una IP supera un umbral definido (ej. 100 peticiones) en un tiempo determinado (ej. 15 minutos), el sistema debe bloquear temporalmente más peticiones de esa IP.			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe registrar cada petición API proveniente de una dirección IP. 2. El sistema debe definir un número máximo de peticiones permitidas (ej. 100). 3. El sistema debe definir una ventana de tiempo para el conteo (ej. 15 minutos). 4. Si una IP supera el límite, el sistema debe rechazar sus siguientes peticiones con un código de error (ej. 429 "Too Many Requests") y un mensaje informativo.		
RESTRICCIONES:			
Este límite debe aplicarse globalmente a todas las rutas de la API para asegurar la estabilidad del servidor.			

HISTORIA DE USUARIO HU-34			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	RNF12	ACTOR	Usuario (Móvil)
NOMBRE DEL REQUERIMIENTO:	Navegación Adaptable (Responsive)		
DESCRIPCIÓN:			
Como usuario, quiero poder usar la aplicación cómodamente desde mi teléfono móvil, asegurando que la navegación y el contenido se ajusten al tamaño de mi pantalla.			
FUNCIONALIDAD:			
El diseño de la interfaz de usuario debe ser "responsive". El menú de navegación principal (sidebar en escritorio) debe colapsar en un menú "hamburguesa" (off-canvas) en dispositivos móviles.			
CRITERIOS DE ACEPTACIÓN:	1. En resoluciones de escritorio (ej. > 992px), el sistema debe mostrar un menú lateral (sidebar) fijo y visible.		

	2. En resoluciones móviles (ej. < 992px), el sistema debe ocultar el menú lateral y mostrar un botón de menú "hamburguesa" en la barra superior. 3. Al hacer clic en el botón "hamburguesa", el sistema debe desplegar un menú lateral (off-canvas) con las mismas opciones de navegación. 4. Las cuadrículas de contenido (ej. la lista de restaurantes en HU-16) deben ajustarse, pasando de múltiples columnas (ej. 3) en escritorio a una sola columna en móvil.
RESTRICCIONES:	
La funcionalidad completa de la aplicación debe estar disponible y ser accesible tanto en escritorio como en móvil.	

HISTORIA DE USUARIO HU-35			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RNF12**	**ACTOR**	**Usuario**
NOMBRE DEL REQUERIMIENTO:	Recibir Retroalimentación de Acciones (Notificaciones)		
DESCRIPCIÓN:			
Como usuario, quiero recibir mensajes claros e inmediatos cuando realizo una acción (exitosa o fallida), para entender qué sucedió sin tener que adivinar.			
FUNCIONALIDAD:			
Implementar un sistema de notificaciones unificado. El sistema debe mostrar "Modales" para errores críticos que detienen al usuario (ej. login fallido) y "Toasts" (ventanas emergentes no intrusivas) para confirmaciones de éxito (ej. registro exitoso).			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe mostrar una notificación "Modal" (que bloquea la pantalla) cuando ocurre un error que requiere la atención del usuario (ej. "Credenciales inválidas", "El correo ya existe"). 2. El sistema debe mostrar una notificación "Toast" (en una esquina, que desaparece sola) cuando una acción se completa con éxito (ej. "Usuario creado correctamente.", "Propuesta enviada."). 3. Los mensajes de error mostrados al usuario deben ser claros y descriptivos (evitando jerga técnica).		
RESTRICCIONES:			
Las notificaciones deben ser consistentes en toda la aplicación.			

HISTORIA DE USUARIO HU-36			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	**RF10**	**ACTOR**	**Sistema**

NOMBRE DEL REQUERIMIENTO:	Cálculo del Ranking Ponderado
DESCRIPCIÓN:	
Como Sistema, quiero recalcular automáticamente el ranking de un restaurante basándome en un algoritmo ponderado (calificación, votos y antigüedad), para asegurar que el ranking sea justo y refleje la opinión actual de la comunidad.	
FUNCIONALIDAD:	
Implementar una lógica de negocio (algoritmo) que se ejecute automáticamente cada vez que una acción afecte las reseñas de un restaurante (crear, editar, eliminar reseña, o votar like/dislike). Este algoritmo debe calcular un nuevo puntaje y actualizarlo en el perfil del restaurante.	
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe recalcular el ranking de un restaurante *siempre* que se cree, actualice o elimine una reseña asociada a él. 2. El sistema debe recalcular el ranking *siempre* que un usuario registre un "like" o "dislike" en una de sus reseñas. 3. El algoritmo debe tomar la calificación base (1-5) de cada reseña. 4. El algoritmo debe sumar o restar valor a esa calificación base según el número de "likes" (suma) y "dislikes" (resta). 5. El algoritmo debe aplicar un factor que dé más peso a las reseñas recientes y menos peso a las antiguas. 6. El puntaje final debe ser un promedio de todas las reseñas ponderadas y debe estar normalizado (ej. siempre entre 1 y 5).
RESTRICCIONES:	
El cálculo del ranking debe ejecutarse dentro de la misma transacción de la base de datos que la acción que lo disparó (ej. crear reseña, votar) para garantizar la consistencia de los datos.	

HISTORIA DE USUARIO HU-37			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	**RNF07**	**ACTOR**	**Sistema**
NOMBRE DEL REQUERIMIENTO:	Validación de Datos de Entrada (Backend)		
DESCRIPCIÓN:			
Como Sistema, quiero validar todos los datos que ingresan a la API (formularios, parámetros de URL), para asegurar la integridad de los datos y prevenir datos corruptos o maliciosos en la base de datos.			
FUNCIONALIDAD:			
Implementar un mecanismo de validación del lado del servidor (antes de que la lógica de negocio se ejecute) para cada dato enviado por un usuario. Debe verificar tipos de datos (numérico, string, email), longitudes y formatos (ej. ID de Mongo).			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe validar todos los datos de entrada para el registro (HU-01) y login (HU-02). 2. El sistema debe validar que los IDs enviados en la URL (ej. para ver un detalle) tengan el formato correcto de Mongo ID.		

	3. El sistema debe validar los datos de todos los formularios (Crear Categoría, Restaurante, Plato, Reseña) [HU-07, HU-10, HU-22, HU-25]. 4. El sistema debe validar que las calificaciones de reseñas sean un número entero entre 1 y 5. 5. Si alguna validación falla, el sistema debe detener la operación y devolver un error 400 (Bad Request) con un mensaje claro que indique qué campo falló.
RESTRICCIONES:	
Esta validación se realiza en el backend y es independiente de las validaciones (HTML5) que pueda tener el frontend.	

HISTORIA DE USUARIO HU-38			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RNF08**	**ACTOR**	**Desarrollador / Admin**
NOMBRE DEL REQUERIMIENTO:	Documentación de API (Explorador de API)		
DESCRIPCIÓN:			
Como Desarrollador (o Admin), quiero poder acceder a una página interactiva que documente todos los endpoints de la API, para entender cómo funciona, qué datos espera y qué responde.			
FUNCIONALIDAD:			
Implementar una herramienta (ej. Swagger/OpenAPI) en el backend que genere automáticamente una documentación interactiva de la API. Esta documentación debe ser accesible desde una ruta específica.			
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe proveer una vista web (UI) para explorar todos los endpoints de la API (GET, POST, PATCH, DELETE). 2. Cada endpoint en la documentación debe especificar qué ruta es, qué método HTTP usa y una descripción de lo que hace. 3. La documentación debe indicar qué endpoints están protegidos (requieren autenticación JWT) y cómo enviar el token. 4. La documentación debe mostrar los "schemas" (DTOs) o ejemplos de los datos que se deben enviar (JSON) y los que se reciben.		
RESTRICCIONES:			
Esta página es una herramienta de desarrollo y puede estar oculta al público general, pero debe ser accesible para el equipo.			

HISTORIA DE USUARIO HU-39			
Prioridad: Baja			
CÓDIGO DEL REQUERIMIENTO:	**RNF12**	**ACTOR**	**Sistema**

NOMBRE DEL REQUERIMIENTO:	Asignación de Imágenes por Defecto (Placeholder)
DESCRIPCIÓN:	
Como Sistema, quiero asignar automáticamente una imagen genérica (placeholder) si un usuario o admin crea un restaurante o un plato sin proporcionar una URL de imagen, para mantener la consistencia visual de la interfaz.	
FUNCIONALIDAD:	
Implementar una lógica en el backend que, al momento de crear un restaurante (HU-10) o un plato (HU-22), verifique si el campo de URL de imagen viene vacío o nulo. Si lo está, el sistema debe rellenar ese campo con una URL de una imagen placeholder estándar.	
CRITERIOS DE ACEPTACIÓN:	1. El sistema debe definir una URL de imagen por defecto para restaurantes. 2. El sistema debe definir una URL de imagen por defecto para platos. 3. Si un usuario propone un restaurante (HU-10) y deja el campo "URL de Imagen" vacío, el sistema debe guardar la URL placeholder del restaurante. 4. Si un admin crea un plato (HU-22) y deja el campo "URL de Imagen" vacío, el sistema debe guardar la URL placeholder del plato.
RESTRICCIONES:	
Esto asegura que la interfaz del frontend nunca muestre imágenes "rotas" o espacios vacíos.	

HISTORIA DE USUARIO HU-40			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	**RNF (Testing)**	**ACTOR**	**Desarrollador**
NOMBRE DEL REQUERIMIENTO:	Poblar Base de Datos con Datos de Prueba (Seed)		
DESCRIPCIÓN:			
Como Desarrollador, quiero poder ejecutar un script que limpie la base de datos y la pueble con un conjunto de datos falsos (usuarios, categorías, restaurantes, reseñas), para probar la aplicación de forma eficiente.			
FUNCIONALIDAD:			
Crear un script ejecutable (ej. seed.js) que se conecte a la base de datos de desarrollo, elimine todos los datos existentes de las colecciones, y cree un conjunto de usuarios (admins, clientes), categorías, restaurantes (aprobados y pendientes) y reseñas.			
CRITERIOS DE ACEPTACIÓN:	1. El script debe poder ejecutarse desde la línea de comandos. 2. El script debe borrar *todos* los datos de las colecciones: usuarios, categorias, restaurantes, platos, reseñas. 3. El script debe crear al menos un usuario "admin" y varios usuarios "cliente". 4. El script debe crear varias categorías. 5. El script debe crear restaurantes, algunos "aprobados" y al menos uno "pendiente".		

	6. El script debe crear reseñas para los restaurantes aprobados y, al finalizar, debe ejecutar el recálculo del ranking (HU-36) para ellos.
RESTRICCIONES:	
Este script solo debe usarse en entornos de desarrollo y nunca debe ejecutarse contra la base de datos de producción.	

3. Metodología

Para el desarrollo del proyecto "My Foodie", se implementó un modelo de gestión ágil que combina el marco de trabajo SCRUM con la metodología visual KANBAN.

Se seleccionó SCRUM como marco principal debido a que el proyecto tiene un alcance definido (los requisitos del taller), pero con una alta complejidad técnica (ej. sistema de ranking transaccional, JS Puro modular). SCRUM nos permitió estructurar el proyecto en Sprints de tiempo fijo, asegurando entregas de valor funcionales y periódicas.

Se complementó con KANBAN para la gestión del flujo de trabajo dentro de cada Sprint. Esto nos proporcionó un tablero visual para gestionar las tareas (Historias de Usuario), limitar el trabajo en progreso e identificar cuellos de botella de forma ágil, lo cual fue crucial para un equipo de desarrollo pequeño.

3.1. Marco de trabajo SCRUM

El proyecto se adhirió a los principios de SCRUM para la organización y gestión del equipo. Esto implicó la definición de roles claros y la ejecución de ceremonias (eventos) específicas para garantizar la comunicación, la transparencia y la adaptación continua.

3.2. Roles del equipo SCRUM

Basado en la estructura de SCRUM, el equipo se organizó de la siguiente manera:

- **Product Owner (PO):**
  - *Sergio Steven Liévano*
  - **Responsabilidad:** Fue el encargado de definir la visión del producto. Su labor principal fue crear, mantener y priorizar el *Product Backlog* (las 40 Historias de Usuario), asegurando que el equipo de desarrollo siempre trabajara en las funcionalidades que aportaban más valor al cumplimiento de los requisitos del taller.
- **Scrum Master (SM):**
  - *Bryan Villabona y Sergio Liévano*
  - **Responsabilidad:** Actuó como facilitador del proceso SCRUM. Se encargó de eliminar impedimentos (técnicos o de gestión), proteger al equipo de

interrupciones externas y asegurar que las ceremonias (eventos) se llevaran a cabo de manera efectiva.

- **Development Team (Equipo de Desarrollo):**
  - *Bryan Villabona (Backend) y Sergio Liévano* (Frontend)
  - **Responsabilidad:** Fue el equipo auto-organizado y multifuncional responsable de construir el *Incremento* del producto en cada Sprint. El equipo tomó las decisiones técnicas sobre cómo implementar la arquitectura del Backend (Node.js, MongoDB Nativo, Transacciones) y del Frontend (JS Puro Modular).

3.3. Eventos SCRUM

El desarrollo se guió por los eventos definidos en SCRUM:

- **Sprint Planning (Planificación del Sprint):** Al inicio de cada Sprint, el equipo completo se reunió. El Product Owner presentaba las Historias de Usuario (HUs) priorizadas del Backlog. El Development Team seleccionaba la cantidad de HUs que se comprometía a completar, creando el *Sprint Backlog*.
- **Daily Standup (Reunión Diaria):** Se realizaron reuniones diarias de 15 minutos para sincronizar al equipo. Cada miembro respondía a las tres preguntas clave: ¿Qué hice ayer? ¿Qué haré hoy? ¿Qué impedimentos tengo?
- **Sprint Review (Revisión del Sprint):** Al finalizar el Sprint, el Development Team presentaba una demostración en vivo del *Incremento* funcional (el software funcionando) al Product Owner y otros interesados. Se validaba que las HUs cumplieran con los Criterios de Aceptación.
- **Sprint Retrospective (Retrospectiva del Sprint):** Inmediatamente después del Review, el equipo (Scrum Master y Dev Team) realizó una reunión interna para reflexionar sobre el Sprint: qué salió bien, qué se podría mejorar, y qué acciones concretas se tomarían para ser más eficientes en el siguiente Sprint.

3.4. Metodología visual KANBAN

Para la gestión de las tareas del *Sprint Backlog*, se utilizó un tablero KANBAN digital en clickup. Este tablero proporcionó visibilidad total del flujo de trabajo.

Las columnas del tablero KANBAN se definieron de la siguiente manera:

1. **Product Backlog:** Contenía todas las HUs (HU-01 a HU-40) priorizadas por el PO.
2. **Sprint Backlog (To Do):** Tareas seleccionadas en el Sprint Planning, listas para ser tomadas.
3. **In Progress (En Progreso):** Tareas que un desarrollador estaba implementando activamente. Se aplicó un límite de "Trabajo en Progreso" (WIP) para fomentar la finalización de tareas antes de iniciar nuevas.

4. **Code Review / Testing (En Revisión):** Tareas completadas por un desarrollador y que estaban siendo validadas por otro miembro del equipo (o el PO) para asegurar la calidad y el cumplimiento de los criterios.
5. **Done (Hecho):** Tareas 100% terminadas, validadas e integradas en la rama principal.

3.5. Fases de desarrollo

El proyecto se dividió en **dos Sprints** principales, permitiendo un desarrollo iterativo e incremental:

Sprint 1: Cimentación de la API y Seguridad (El Núcleo del Backend)

- **Objetivo del Sprint:** Establecer la arquitectura completa del backend, la conexión a la base de datos (MongoDB Nativo), el sistema de seguridad (JWT) y el primer CRUD administrativo.
- **Historias de Usuario Clave (Seleccionadas del Backlog):**
  - HU-01 (Registro), HU-02 (Login), HU-03 (Logout).
  - HU-04 (Protección de Vistas - Frontend), HU-05 (Protección de Funciones - Backend).
  - HU-06 (Ver Categorías), HU-07 (Crear Categoría), HU-08 (Editar Categoría), HU-09 (Eliminar Categoría).
  - HU-33 (Rate Limiting), HU-37 (Validación de Datos), HU-38 (Documentación API), HU-40 (Script Seed).
- **Resultado (Incremento Funcional):** Al final del Sprint 1, se entregó una API REST funcional y segura. Permitía el registro y login de usuarios, diferenciaba roles "admin" y "cliente", y ofrecía un CRUD completo y protegido para "Categorías". La API estaba documentada en Swagger y lista para ser consumida.

Sprint 2: Lógica de Negocio, Sistema de Ranking y Frontend (La Aplicación Completa)

- **Objetivo del Sprint:** Desarrollar el corazón de la lógica de negocio (Restaurantes, Platos, Reseñas, Ranking Transaccional) y construir la totalidad de la interfaz de usuario en JavaScript Puro.
- **Historias de Usuario Clave (Seleccionadas del Backlog):**
  - HU-10 (Proponer Restaurante - Cliente), HU-11/12/13 (Flujo de Aprobación - Admin).
  - HU-14/15 (CRUD de Restaurantes - Admin).
  - HU-16/17/18/19 (Explorar Restaurantes - Cliente).
  - HU-20 (Ver Detalle), HU-21/22/23/24 (CRUD de Platos).
  - HU-25/26/27/28 (CRUD de Reseñas).
  - HU-29/30 (Votación Like/Dislike).
  - HU-31/32 (Dashboard y Mis Reseñas).
  - HU-36 (Lógica de Ranking Transaccional).
  - HU-34 (Responsive), HU-35 (Notificaciones).

- **Resultado (Incremento Funcional):** Al final del Sprint 2, se entregó una aplicación Full-Stack 100% funcional. El frontend (JS Puro) consumía la API del backend, los usuarios podían registrarse, proponer restaurantes, reseñar y votar. Los administradores podían gestionar el contenido. El sistema de ranking ponderado funcionaba automáticamente con cada interacción, asegurando la integridad de los datos mediante transacciones.

4. Tecnologías implementadas

La arquitectura del proyecto "My Foodie" se basa en una separación completa de responsabilidades entre un Backend (API REST) y un Frontend (Cliente Web), cada uno construido con un stack tecnológico específico para cumplir con los requisitos del taller.

4.1. Backend (API Rest)

El servidor se construyó sobre el ecosistema Node.js, priorizando la eficiencia, la seguridad y el control directo sobre la base de datos, tal como lo exigen los requerimientos.

Tecnología	Versión (Aprox.)	Justificación de Uso (Implementación)
Node.js	v18+	Entorno de ejecución de JavaScript del lado del servidor. Base de toda la aplicación backend.
Express.js	v4.19.2	Framework web minimalista para Node.js. Se utilizó para la creación del servidor, la gestión de rutas (enrutamiento) y la configuración de middlewares.
MongoDB (Driver Nativo)	v6.6.1	Driver oficial de MongoDB para Node.js. Se utilizó para cumplir con el requisito de interactuar directamente con la base de datos y permitir la implementación manual de **Transacciones** (session.withTransaction()), asegurando la integridad de los datos en operaciones críticas.
JSON Web Token (jsonwebtoken)	v9.0.2	Utilizado para generar (.sign()) los tokens de acceso seguros (JWT) durante el inicio de sesión del usuario.

Tecnología	Versión (Aprox.)	Justificación de Uso (Implementación)
Passport (passport-jwt)	v4.0.1	Estrategia de Passport.js para la autenticación basada en JWT. Se usó para crear el middleware autenticar, protegiendo los endpoints de la API al validar los tokens enviados en el header Authorization.
Bcrypt.js	v5.1.1	Librería para el hashing de contraseñas. Se utilizó para encriptar (.hash()) las contraseñas de los usuarios antes de guardarlas en la base de datos y para compararlas (.compare()) durante el login.
Express-Validator	v7.0.1	Middleware de validación de datos. Se utilizó para definir "DTOs" (Data Transfer Objects) y validar todos los datos de entrada (body, params) en las rutas, asegurando la integridad de los datos antes de que lleguen a los servicios.
Express-Rate-Limit	v7.1.5	Middleware de seguridad. Se implementó globalmente en el servidor (server.js) para limitar el número de peticiones por IP, previniendo ataques de fuerza bruta o de denegación de servicio (DoS).
Swagger-UI-Express	v5.0.0	Generador de documentación de API interactiva (OpenAPI). Se utilizó para crear la ruta /api-docs, proveyendo una interfaz web para visualizar y probar todos los endpoints de la API.
Dotenv	v16.3.1	Utilizado para cargar variables de entorno (como MONGO_URI, JWT_SECRET, PORT) desde un archivo .env al proceso de Node.js, manteniendo las credenciales seguras y fuera del código fuente.

Tecnología	Versión (Aprox.)	Justificación de Uso (Implementación)
CORS	v2.8.5	Middleware para habilitar el Intercambio de Recursos de Origen Cruzado. Esencial para permitir que el Frontend (ej. http://127.0.0.1:5500) se comunique con el Backend (ej. http://localhost:4000).
Nodemon	v3.0.1	Herramienta de desarrollo que reinicia automáticamente el servidor de Node.js cuando detecta cambios en el código, agilizando el ciclo de desarrollo.

4.2. Frontend (Cliente Web)

El cliente web se desarrolló cumpliendo el requisito estricto de no utilizar frameworks de JavaScript (React, Vue, Angular). Toda la interactividad, modularidad y gestión de estado se construyó con JavaScript puro (Vanilla JS) y tecnologías web estándar.

Tecnología	Versión (Aprox.)	Justificación de Uso (Implementación)
HTML5	N/A	Lenguaje de marcado para la estructuración semántica de todas las vistas de la aplicación (ej. index.html, dashboard.html, detalle.html).
CSS3	N/A	Lenguaje de hojas de estilo. Se utilizó en style.css para definir estilos personalizados, aplicar la paleta de colores de "My Foodie" y ajustar componentes de Bootstrap.
JavaScript (ES6+ Modules)	N/A	Lenguaje de programación del cliente. Se utilizó en su versión moderna (ES Modules) para toda la lógica de la aplicación. Se estructuró el código en módulos (components, pages, services, utils) para la manipulación del DOM, gestión de

Tecnología	Versión (Aprox.)	Justificación de Uso (Implementación)
		eventos, enrutamiento del lado del cliente (main.js) y consumo de la API (fetch).
Bootstrap	v5.3.3	Framework de CSS/UI. Se utilizó para el diseño responsive (Grid, Off-canvas) y para los componentes de interfaz de usuario (Botones, Modales, Tarjetas, Formularios, Navs), permitiendo un desarrollo rápido de una UI limpia y adaptable.
Bootstrap Icons	v1.11.3	Biblioteca de iconos SVG. Se utilizó para toda la iconografía de la aplicación (ej. iconos de navegación, estrellas de calificación, botones de like/dislike).

4.3. Herramientas de desarrollo y gestión

Herramienta	Uso
Git / GitHub	Sistema de control de versiones (CVS) y plataforma de repositorios para el trabajo colaborativo y la gestión del código fuente (Backend y Frontend).
KANBAN	Metodología visual (implementada en ClickUp) para la gestión de tareas del Sprint, limitando el trabajo en progreso y visualizando el flujo.
Visual Studio Code	Entorno de Desarrollo Integrado (IDE) principal para la codificación del proyecto.

5. Evidencia de Planteamiento de Plataforma de Trabajo.

A continuación, se documenta la evidencia tangible del trabajo colaborativo, la gestión del proyecto y el cumplimiento de las ceremonias ágiles, tal como se describió en la sección de Metodología.

5.1. Repositorios del Proyecto (GitHub)

El proyecto se gestionó bajo un sistema de control de versiones (Git) y se alojó en dos repositorios de GitHub separados, cumpliendo con el requerimiento de desacoplamiento (Backend y Frontend). En ellos se evidencia el trabajo colaborativo, el uso de ramas de desarrollo y el seguimiento mediante *conventional commits*.

- **Link del Repositorio Backend (My Foodie API):**

  https://github.com/BryanVillabona/My_Foodie_Backend.git

- **Link del Repositorio Frontend (My Foodie Client):**

  https://github.com/sergiosteven66/My_Foodie_Frontend.git

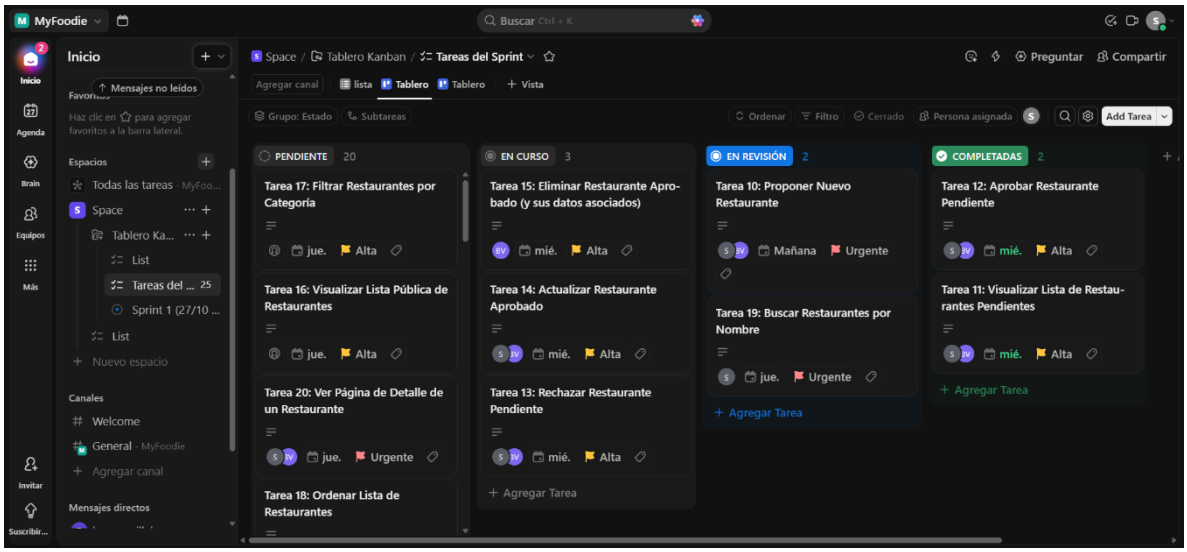
5.2. Evidencia del Tablero Scrum (KANBAN)

Para la gestión del flujo de trabajo diario y la visualización de las Historias de Usuario, se utilizó un tablero KANBAN (en ClickUp). A continuación, se presentan capturas de pantalla que evidencian el *Product Backlog* (las 40 HUs), el *Sprint Backlog* y el flujo de tareas por las columnas KANBAN.

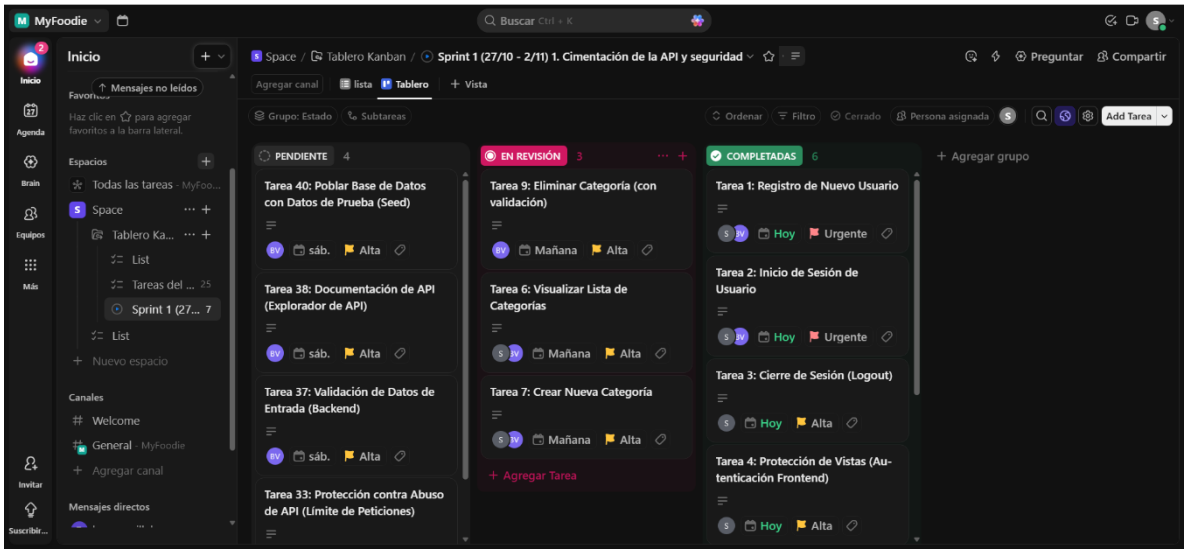
- **Link Público al Tablero KANBAN:** https://sharing.clickup.com/90132680511/b/h/2ky573tz-193/299732beabc5497

- **Link Público al tablero de Sprint:** https://sharing.clickup.com/90132680511/b/h/6-901321961496-2/9af1f9e3326c98d

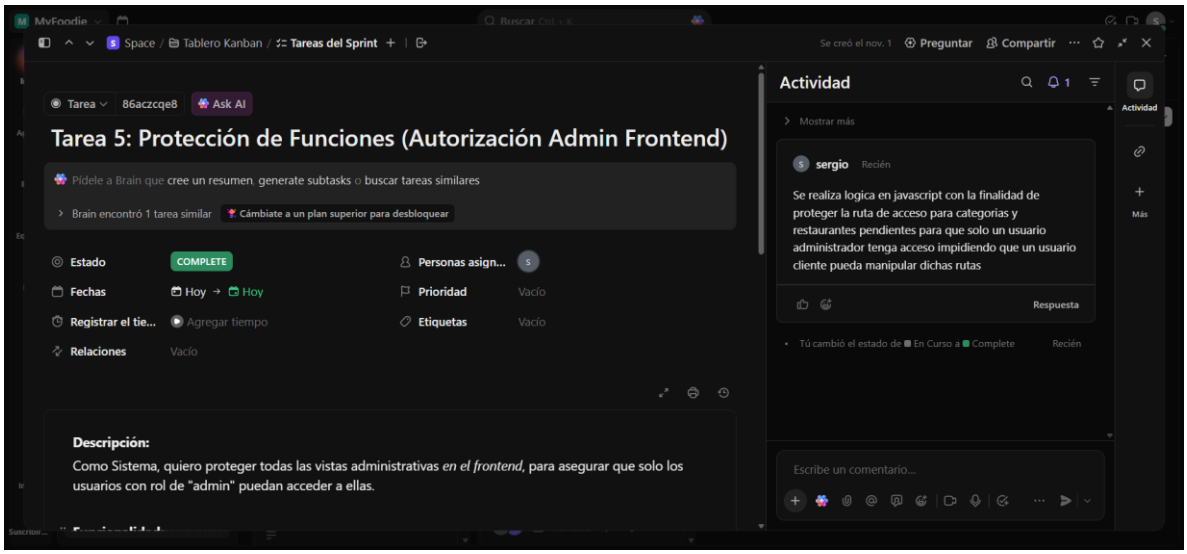
- **Captura 1: Vista General del Product Backlog**



- **Captura 2: Vista del Tablero KANBAN (Sprint Activo)**



- **Captura 3: Detalle de una Tarea (Historia de Usuario)**



6. Documentación de los resultados y funcionamiento del producto final

El resultado de este proyecto es una aplicación web full-stack, "My Foodie", completamente funcional. El producto cumple con la totalidad de los requerimientos funcionales y no funcionales, implementando con éxito una arquitectura desacoplada que consta de dos componentes principales:

1. **Backend (API REST):** Una API segura y robusta construida con Node.js y Express.
2. **Frontend (Cliente Web):** Una aplicación de una sola página (SPA) responsiva, construida íntegramente con HTML, CSS y JavaScript Puro (Vanilla JS).

A continuación, se detalla el funcionamiento interno de cada componente y cómo interactúan para cumplir con las tareas planteadas.

6.1. Funcionamiento del Backend (API REST)

El backend actúa como el cerebro de la aplicación, gestionando toda la lógica de negocio, la seguridad y la persistencia de los datos.

- **Arquitectura Modular:** El servidor está estructurado en capas claras de responsabilidad. Las **Rutas** (/routes) definen los endpoints y aplican los middlewares. Los **Controladores** (/controllers) manejan las peticiones (request) y respuestas (response). Los **Servicios** (/services) contienen la lógica de negocio y las operaciones de base de datos. Los **DTOs** (/dtos) definen las reglas de validación para los datos de entrada.

- **Seguridad y Autenticación (JWT):** El funcionamiento de la seguridad se basa en JSON Web Tokens.

    1. Un usuario envía credenciales (HU-02), el servicio auth.services.js las valida contra la base de datos usando bcrypt.

    2. Si son válidas, se genera un JWT firmado (jsonwebtoken) que contiene el ID del usuario, su nombre y su rol ("cliente" o "admin").

    3. Para cualquier solicitud a una ruta protegida (ej. crear una reseña), el middleware autenticar (auth.middleware.js) intercepta la petición.

    4. Utilizando passport-jwt, el middleware valida la firma del token, verifica que no haya expirado y extrae el *payload* (los datos del usuario), adjuntándolos a la petición para su uso posterior.

- **Autorización Basada en Roles:** Para funciones críticas (ej. aprobar restaurantes, crear categorías), se utiliza un segundo middleware, autorizarAdmin. Este middleware (ejecutado después de autenticar) comprueba si el rol del usuario (obtenido del token) es "admin". Si no lo es, deniega el acceso con un error 403 (Prohibido), impidiendo que un "cliente" acceda a funciones administrativas.

- **Gestión de Base de Datos (MongoDB Nativo y Transacciones):** En cumplimiento con los requisitos, el proyecto *no utiliza Mongoose*. Toda la interacción con la base de datos se realiza a través del **driver nativo de MongoDB** (mongodb).

La funcionalidad más crítica del backend es el uso de **Transacciones de MongoDB**. Esto garantiza la integridad de los datos (principio de atomicidad) en operaciones complejas.

Ejemplo (Flujo de Creación de Reseña - HU-25):

1. Un cliente envía una reseña (calificación y comentario).

2. El servicio reseñas.services.js inicia una sesión transaccional (cliente.startSession()).

3. Dentro de la transacción, el sistema realiza dos operaciones: a. Inserta la nueva reseña en la colección reseñas. b. Llama a la función recalcularRanking (ranking.js).

4. La función recalcularRanking (que también se ejecuta *dentro* de la sesión) consulta todas las reseñas del restaurante, aplica el algoritmo ponderado (HU-36) y actualiza (updateOne) el campo rankingPonderado en la colección restaurantes.

5. Si *ambas* operaciones (insertar reseña y actualizar ranking) tienen éxito, la transacción se confirma (commit).

6. Si *alguna* de las dos falla (ej. la actualización del ranking falla), la transacción se anula (abort) y la creación de la reseña se revierte (rollback), impidiendo que existan reseñas que no se reflejen en el ranking.

Este mismo principio transaccional se aplica al votar (HU-29, HU-30) y al eliminar un restaurante (HU-15), donde se borran el restaurante, sus platos y sus reseñas en una sola operación atómica.

6.2. Funcionamiento del Frontend (Cliente JavaScript Puro)

El frontend se desarrolló como una Aplicación de Página Única (SPA) simulada, utilizando exclusivamente **HTML, CSS y JavaScript Puro (Vanilla JS)**, cumpliendo con el requisito principal del taller.

- **Arquitectura Modular (ES Modules):** A pesar de no usar un framework, el código JavaScript está altamente modularizado (similar a la arquitectura del backend):
    - services/: Contiene toda la lógica de fetch para consumir la API del backend.
    - components/: Módulos que generan piezas de HTML reutilizables (ej. Layout.js crea la barra de navegación, RestauranteCard.js crea las tarjetas de restaurante).
    - pages/: Contienen la lógica específica que se ejecuta para cada vista (ej. detalle.js orquesta la carga de datos del restaurante y sus reseñas).
    - utils/: Funciones de ayuda para autenticación (auth.js), protección de rutas (guards.js) y renderizado (rendering.js).
- **Enrutamiento y Guardas de Ruta (Routing & Guards):** El archivo main.js actúa como el controlador de enrutamiento principal. En cada carga de página:
    1. Verifica si el usuario está en la página de login (/index.html) o en una página interna.
    2. Llama a getToken() de auth.js para ver si existe un token en localStorage.
    3. Aplica las guardas de ruta (HU-05): Si un usuario sin token intenta ver /dashboard.html, es redirigido a /index.html. Si un usuario *con* token intenta ver /index.html, es redirigido a /dashboard.html.

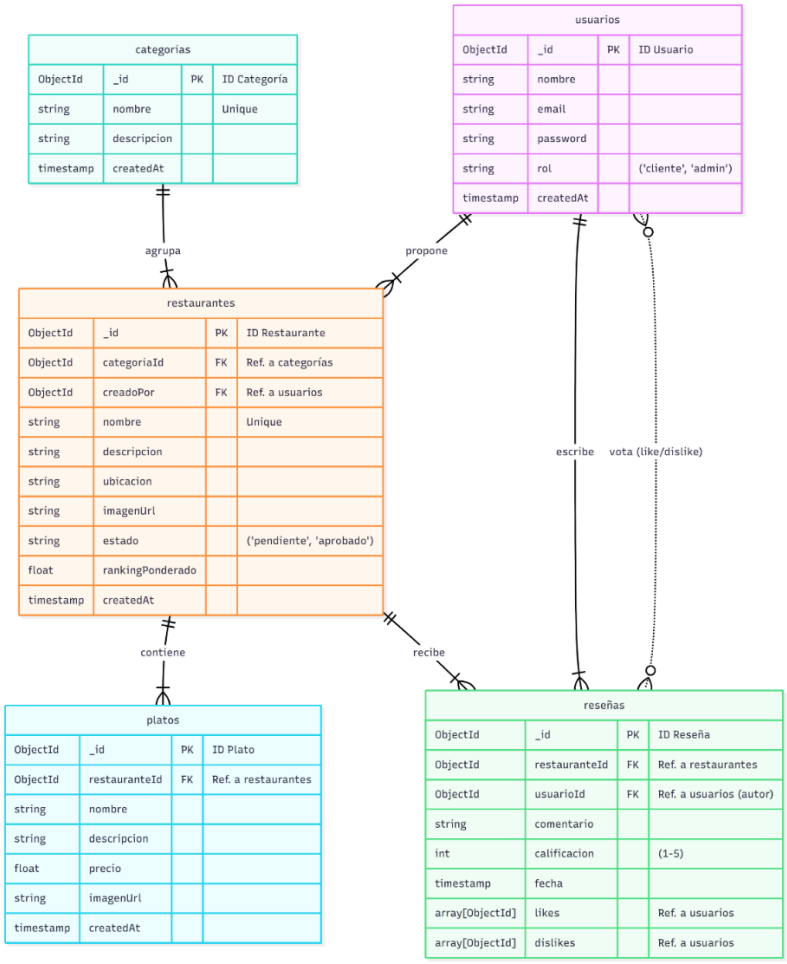
4. Finalmente, importa dinámicamente (await import(...)) solo el script de la página correspondiente (ej. initDashboardPage()).

- **Gestión de Roles y UI Dinámica:** La interfaz de usuario se renderiza dinámicamente según el rol del usuario. Al cargar el layout, se obtiene el rol desde el token (getUserData() en auth.js).

Ejemplo (Flujo de Admin en Detalle - HU-14, HU-22):

1. Un usuario "admin" visita detalle.html.
2. detalle.js detecta que currentUserRole es "admin".
3. Al renderizar el HTML, el sistema *sí* incluye los botones "Editar Restaurante", "Eliminar Restaurante" y "Añadir Plato".
4. Si un usuario "cliente" visita la misma página, esas variables de rol son falsas y los botones no se añaden al DOM, ocultando la funcionalidad.

6.3. Modelo de la base de datos



Este diagrama muestra las 5 colecciones principales:

1. **usuarios**: Almacena los datos de registro, incluyendo el rol.
2. **categorias**: Lista simple de categorías gestionada por el admin.
3. **restaurantes**: La entidad central.
    - Se conecta 1:N con categorias (un restaurante tiene una categoría).
    - Se conecta 1:N con usuarios (un usuario creadoPor el restaurante).
4. **platos**:
    - Se conecta 1:N con restaurantes (un plato pertenece a un solo restaurante).
5. **reseñas**: La entidad más conectada.
    - Se conecta 1:N con restaurantes (una reseña es para un solo restaurante).
    - Se conecta 1:N con usuarios (una reseña tiene un solo autor).
    - Implementa la relación N:M para los votos, almacenando los _id de los usuarios que han dado like o dislike en arrays.

6.4. Cumplimiento de Tareas Clave

El producto final cumple con todas las tareas planteadas, destacando:

- **Backend y Frontend en repositorios separados.**
- **Backend 100% en Node.js + Express + MongoDB Nativo**.
- **Frontend 100% en HTML + CSS + JavaScript Puro**.
- **Autenticación JWT completa (Auth + Roles)**.
- **Uso de Transacciones en MongoDB** para la lógica de negocio (ranking).
- **Validaciones robustas (express-validator)** y **Documentación (swagger)** en el backend.
- **Interfaz de usuario responsive y amigable**.

7. Conclusiones

7.1. Conclusiones generales del proyecto

Tras la finalización de los dos Sprints de desarrollo, se concluye que el proyecto "My Foodie" ha sido implementado exitosamente, cumpliendo con la totalidad de los requerimientos funcionales y no funcionales establecidos en el taller.

1. **Éxito en el Desafío Técnico del Stack:** El proyecto demuestra la viabilidad de construir una aplicación full-stack moderna y robusta adhiriéndose a restricciones técnicas significativas.
    - En el **Backend**, se logró implementar una API segura y eficiente utilizando exclusivamente el **driver nativo de MongoDB**, prescindiendo de ODMs como Mongoose. Esto forzó un entendimiento más profundo de las operaciones de base de datos, especialmente en la implementación manual de **Transacciones**, que fueron cruciales para garantizar la integridad de los datos en el sistema de ranking (HU-36).
    - En el **Frontend**, se superó el desafío de **no utilizar frameworks** (React, Vue, Angular). Se construyó una arquitectura de JavaScript Puro (Vanilla JS) que es modular, escalable y mantenible, utilizando componentes, servicios y un enrutador del lado del cliente, demostrando un dominio de las tecnologías web fundamentales.
2. **Solidez de la Lógica de Negocio:** El sistema de ranking ponderado (HU-36), que representa el núcleo de la lógica de negocio, se implementó exitosamente. El algoritmo (ranking.js) se integra de forma atómica (transaccional) con todas las interacciones del usuario (reseñas y votos), asegurando que el ranking sea un reflejo fiable y en tiempo real de la actividad de la comunidad, cumpliendo así con la solución a la situación problema.
3. **Calidad del Producto Final:** La aplicación resultante es segura, funcional y usable. La seguridad está garantizada por un flujo completo de autenticación y autorización basado en JWT (HU-01, HU-04, HU-05), validaciones de entrada (express-validator) (HU-37) y protección de API (express-rate-limit) (HU-33). La usabilidad se logra con una interfaz limpia y responsive (HU-34) y un sistema claro de notificaciones al usuario (HU-35).
4. **Efectividad de la Metodología:** La combinación de **SCRUM** (para la gestión de Sprints y eventos) y **KANBAN** (para la gestión visual del flujo de trabajo) fue una decisión acertada. Permitió al equipo tener un plan estructurado (Sprints) pero con la

flexibilidad de adaptar las tareas diarias (KANBAN), lo cual fue esencial para abordar la alta carga técnica del Sprint 2.

7.2. Conclusiones de la Reunión de Retrospectiva del Sprint

Al finalizar el proyecto, el equipo de desarrollo (Development Team) y el Scrum Master llevaron a cabo la Sprint Retrospective final, con el objetivo de identificar puntos de mejora para futuros proyectos. Las conclusiones fueron las siguientes:

- **Qué salió bien:** La comunicación dentro del equipo fue constante y fluida. La decisión de pivotar rápidamente (ej. cambiar paleta de colores) basándose en la retroalimentación interna fue un gran acierto.
- **Qué se podría mejorar:** La estimación inicial del tiempo para algunas tareas fue demasiado optimista. Para futuros proyectos, se podría dedicar más tiempo a la fase de diseño simple antes de la codificación.
- **Acciones a tomar:** Implementar un estándar de nombrado de ramas en Git más estricto desde el inicio del próximo proyecto para mejorar aún más la organización del repositorio.