FOODIE RANK

FABIAN CAMILO PERTUZ TORRES Y CARLOS MARIO VILLAMIZAR MEDINA

U1

PEDRO GOMEZ

CAMPUSLANDS
SALON ARTEMIS
RUTA NODEJS
FLORIDABLANCA
2025

FOODIE RANK

1. SITUACIÓN PROBLEMA

En la actualidad, los usuarios buscan constantemente experiencias gastronómicas nuevas y desean compartir sus opiniones con otros comensales. Sin embargo, muchas plataformas carecen de un sistema transparente, equilibrado y justo que combine calificaciones, reseñas y popularidad en un solo ranking confiable.

Foodie Rank surge como una solución a esta necesidad. Es una aplicación web full-stack desarrollada en **Node.js + Express** (backend) y **HTML, CSS y JavaScript puro** (frontend), que permite a los usuarios **registrar, calificar y rankear restaurantes y platos** de manera dinámica y segura.

El proyecto busca ofrecer una experiencia interactiva en la que los usuarios puedan dejar reseñas auténticas, gestionar sus perfiles y descubrir nuevos lugares recomendados por la comunidad.
Además, permite a los administradores **gestionar categorías, aprobar restaurantes y moderar reseñas**, garantizando calidad y confiabilidad en la información publicada.

Con esto, **Foodie Rank** pretende cubrir una oportunidad de mercado en la que convergen la gastronomía, la comunidad y la tecnología, ofreciendo una herramienta moderna, escalable y justa para la evaluación culinaria.

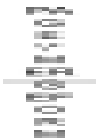
presentado, razón por la cual se espera una descripción más ampliada y soportada en referentes.

2. LEVANTAMIENTO DE REQUERIMIENTOS

3. REQUERIMIENTOS.

3.1. Requerimientos Funcionales

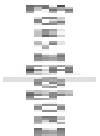
Código	Requerimiento	Descripción
RF01	**Registro e inicio de sesión de usuarios**	**El sistema debe permitir registrar nuevos usuarios y autenticar mediante JWT.**
RF02	**Roles de usuario y administrador**	**El sistema debe diferenciar permisos entre usuarios comunes y administradores.**
RF03	**CRUD de restaurantes**	**Los administradores pueden crear, aprobar, editar y eliminar restaurantes.**



RF04	**CRUD de platos**	**Los usuarios y administradores pueden crear, editar y eliminar platos asociados a un restaurante.**
RF05	**Gestión de reseñas**	**Los usuarios pueden crear, editar o eliminar reseñas con comentarios y calificaciones (1–5 estrellas).**
RF06	**Likes/Dislikes en reseñas**	**Los usuarios pueden dar like o dislike a reseñas de otros.**
RF07	**Ranking de restaurantes**	**El sistema debe calcular un ranking ponderado basado en calificaciones, likes/dislikes y fecha.**
RF08	**CRUD de categorías**	**Los administradores pueden crear, editar y eliminar categorías gastronómicas.**
RF09	**Documentación de API**	**Todos los endpoints deben estar documentados con Swagger.**
RF10	**Listado y filtrado**	**Los usuarios pueden listar restaurantes y filtrarlos por categoría o ranking.**

3.2. Requerimientos no funcionales

Código	Requerimiento	Descripción
RNF01	**Seguridad**	**Implementar autenticación JWT, cifrado de contraseñas con bcrypt y control de peticiones con rate limiting.**
RNF02	**Escalabilidad**	**Arquitectura modular (routes, controllers, models, middlewares, services).**
RNF03	**Rendimiento**	**La API debe responder en menos de 2 segundos para solicitudes estándar.**

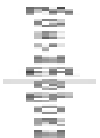


RNF04	**Usabilidad**	**Interfaz intuitiva, responsiva y sin frameworks (HTML, CSS y JS puro).**
RNF05	**Disponibilidad**	**Sistema estable, con control de errores y manejo de excepciones.**
RNF06	**Consistencia de datos**	**Uso de transacciones en MongoDB para operaciones críticas.**
RNF07	**Documentación**	**API documentada con Swagger y README detallado.**
RNF08	**Versionado**	**API versionada bajo semver (ej. v1.0.0).**

4. HISTORIAS DE USUARIO CON CRITERIOS DE ACEPTACIÓN

1.

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF01	**Actor**	Usuario
NOMBRE DEL REQUERIMIENTO	**Registro e inicio de sesión**		
Descripción			
Como usuario, quiero poder registrarme e iniciar sesión para acceder a la plataforma y dejar reseñas sobre restaurantes y platos.			
Funcionalidad			
Permitir a usuarios registrarse con correo y contraseña, validar campos obligatorios, cifrar la contraseña con bcrypt, iniciar sesión y recibir un token JWT que habilite acciones autenticadas en la API.			
Criterios de aceptación	1. El sistema permite crear una cuenta única con correo válido; al registrarse la contraseña se guarda cifrada. 2. Al iniciar sesión con credenciales válidas el sistema devuelve un JWT válido y la información básica del usuario. y 3. Los intentos de registro con correo duplicado son rechazados con un error claro. 4. Los intentos de login con credenciales erradas retornan error 401 con mensaje claro.		
Restricciones			



No se debe permitir el registro con correos duplicados; solo usuarios autenticados con token válido pueden acceder a rutas protegidas.

2.

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF02	**Actor**	Administrador
NOMBRE DEL REQUERIMIENTO	**Roles de usuario y administrador**		
Descripción			
Como administrador, quiero gestionar los roles de usuario para definir los permisos y controlar el acceso a las funciones del sistema.			
Funcionalidad			
Implementar un sistema de roles (ej. user, admin) y middleware de autorización que verifique permisos en rutas críticas (aprobación, gestión de categorías, eliminación definitiva). Proveer endpoint para que administradores cambien roles de usuarios y visualizar rol actual.			
Criterios de aceptación	1. El sistema distingue entre user y admin y aplica permisos en endpoints. 2. Solo los administradores pueden invocar endpoints de administración. 3. Intentos de acceso a endpoints restringidos por usuarios no admin retornan 403.		
Restricciones			
Solo administradores autorizados pueden asignar o modificar roles; las modificaciones se registran en logs.			

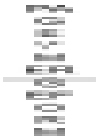
3.

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF03	**Actor**	Administrador
NOMBRE DEL REQUERIMIENTO	**CRUD de restaurantes**		
Descripción			
Como administrador, quiero crear, aprobar, editar y eliminar restaurantes para mantener actualizada la base de datos de lugares disponibles.			
Funcionalidad			

Permitir a usuarios proponer restaurantes mediante endpoint público , almacenar propuestas en estado pending, y ofrecer a administradores endpoints para aprobar , editar y eliminar restaurantes. Validar unicidad de nombre y datos requeridos (nombre, ubicación, categoría).	
Criterios de aceptación	1. Propuestas de usuarios quedan en estado pending y no se muestran en listados públicos hasta ser aprobadas. 2. Los administradores pueden aprobar o rechazar propuestas; al aprobar, el restaurante cambia a approved. 3. El sistema valida que no existan nombres duplicados antes de crear/aprobar. 4. Administradores pueden editar datos y eliminar restaurantes vía endpoints protegidos.
Restricciones	
Solo administradores pueden aprobar/rechazar/Eliminar restaurantes; restaurantes no aprobados no aparecen en listados públicos.	

4.

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF04	**Actor**	Usuario / Administrador
NOMBRE DEL REQUERIMIENTO	**CRUD de platos**		
Descripción			
Como usuario, quiero agregar platos a un restaurante para compartir su descripción y calificación con otros usuarios.			
Funcionalidad			
Proveer endpoints para crear , leer, actualizar y eliminar platos vinculados a un restaurante. Validar existencia del restaurante, unicidad del nombre del plato dentro del mismo restaurante, y campos mínimos (nombre, descripción, precio opcional). Los Administradores pueden moderar o eliminar platos que infrinjan normas.			
Criterios de aceptación	1. Se puede crear un plato solo si el restaurante existe y está aprobado. 2. No se permite crear dos platos con el mismo nombre dentro del mismo restaurante. 3. El autor del plato puede editar o eliminar su plato; el administrador puede eliminar cualquier plato. 4. Endpoints retornan códigos HTTP correctos (201 al crear, 200 al		



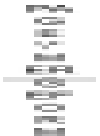
	actualizar, 404 si no existe).
Restricciones	
No se aceptan platos asociados a restaurantes no aprobados o inexistentes.	

5.

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF05	**Actor**	Usuario
NOMBRE DEL REQUERIMIENTO	**Gestión de reseñas**		
Descripción			
Como usuario, quiero crear, editar o eliminar reseñas sobre platos o restaurantes para compartir mi experiencia con otros usuarios.			
Funcionalidad			
Implementar endpoints para crear , editar y eliminar reseñas; cada reseña incluirá comentario, calificación (1–5), referencia a restaurante o plato, autor y fecha. Al crear o eliminar reseñas se debe actualizar promedio y estadísticas usando transacción en MongoDB para mantener consistencia.			
Criterios de aceptación	1. El usuario autenticado puede crear una reseña con calificación entre 1–5. 2. El autor puede editar o eliminar su reseña; otros usuarios no pueden. 3. El promedio de calificaciones y conteos se actualizan inmediatamente tras transacciones exitosas. 4. No se permite más de una reseña por mismo usuario sobre el mismo recurso (plato o restaurante).		
Restricciones			
Un usuario no puede dejar múltiples reseñas para el mismo plato/restaurante; operaciones críticas (crear + actualizar promedio) se ejecutan en transacción.			

6.

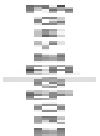
HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL	RF06	**Actor**	Usuario



REQUERIMIENTO:			
NOMBRE DEL REQUERIMIENTO	**Likes & dislikes en reseñas**		
Descripción			
Como usuario, quiero poder dar like o dislike a reseñas de otros para destacar las más útiles y relevantes.			
Funcionalidad			
Agregar endpoints para reaccionar a reseñas permitiendo like, dislike o remover reacción; almacenar reacciones por usuario para prevenir duplicados; actualizar conteo de reacciones y validar que no se reaccione a reseñas propias.			
Criterios de aceptación	1. Usuario autenticado puede dar like o dislike a una reseña ajena. 2. No se permite dar más de una reacción del mismo tipo a la misma reseña por el mismo usuario; se puede cambiar o remover la reacción. 3. Conteos de likes/dislikes se actualizan atómica y consistentemente. 4. Intentos de reaccionar a reseñas propias retornan error 400/403.		
Restricciones			
Solo usuarios autenticados pueden reaccionar; reacciones se registran por usuario para evitar duplicados.			

7.

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF07	**Actor**	Usuario
NOMBRE DEL REQUERIMIENTO	**Ranking de restaurantes**		
Descripción			
Como usuario, quiero ver un ranking de restaurantes basado en calificaciones, likes/dislikes y antigüedad de reseñas para descubrir los mejores lugares.			
Funcionalidad			
Implementar un algoritmo que calcule un puntaje ponderado por restaurante con factores: promedio de calificaciones, volumen y calidad (likes/dislikes) y reciente actividad (fecha de reseñas). Proveer endpoint que devuelva lista ordenada por puntaje y permitir parámetros de paginación y filtros por categoría.			
Criterios de aceptación	1. El endpoint de ranking retorna restaurantes ordenados por puntaje calculado. 2. El puntaje incorpora calificación promedio, ajuste por likes/dislikes y penalización por reseñas muy antiguas (configurable).		



	3. Ranking se actualiza cuando se agregan/editar/eliminan reseñas (o reacciones). 4. Se soportan filtros por categoría y paginación.
Restricciones	
Solo se consideran restaurantes aprobados; el algoritmo debe ser determinista y documentado en README.	

8.

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF08	**Actor**	Administrador
NOMBRE DEL REQUERIMIENTO	**CRUD de categorías**		
Descripción			
Como administrador, quiero crear, editar y eliminar categorías gastronómicas para organizar mejor los restaurantes y platos.			
Funcionalidad			
Proveer endpoints protegidos para crear, listar, actualizar y eliminar categorías. Validar unicidad del nombre de categoría; al eliminar una categoría reasignar restaurantes afectados a sin categoria o manejar migración.			
Criterios de aceptación	1. Los administradores pueden crear categorías únicas (nombre). 2. Editar y eliminar categorías es posible mediante endpoints protegidos. 3. Al eliminar una categoría, los restaurantes vinculados quedan en sin categoría o se reasignan según política. 4. La lista pública de categorías está disponible para el frontend.		
Restricciones			
Solo administradores pueden gestionar categorías; no se permiten nombres duplicados.			

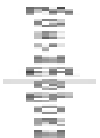
9.

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF09	**Actor**	Usuario / Desarrollador

NOMBRE DEL REQUERIMIENTO	Documentación de API
Descripción	
Como desarrollador, quiero que la API esté documentada con Swagger para poder probar y consumir los endpoints fácilmente.	
Funcionalidad	
Configurar swagger-ui-express para exponer documentación interactiva en /api docs, describiendo todos los endpoints (método, ruta, parámetros, cuerpos, respuestas y códigos HTTP). Incluir ejemplos y esquemas (OpenAPI) y excluir datos sensibles.	
Criterios de aceptación	1. /api-docs muestra documentación actualizada y navegable de todos los endpoints. 2. Cada endpoint documentado incluye parámetros, ejemplos de request/response y códigos HTTP. 3. No se exponen secretos o variables de entorno en la documentación. 4. La documentación refleja la versión API (semver) y el changelog básico.
Restricciones	
No incluir claves privadas ni datos sensibles en ejemplos de Swagger; solo datos de prueba.	

10.

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF10	**Actor**	Usuario
NOMBRE DEL REQUERIMIENTO	**Listado y filtrado de restaurantes**		
Descripción			
Como usuario, quiero listar y filtrar restaurantes por categoría, popularidad o ranking para encontrar lugares de mi preferencia.			
Funcionalidad			
Implementar endpoint público que retorne listado con opciones de filtro por categoría, orden por ranking/popularidad/nombre y paginación. Cada ítem muestra promedio de calificación, número de reseñas, categoría y ubicación.			
Criterios de aceptación	1. El endpoint devuelve restaurantes aprobados con paginación. 2. Filtros por categoría y ordenamiento por ranking o nombre funcionan correctamente. 3. Cada resultado incluye información mínima (nombre, calificación promedio, categoría, ubicación). 4. Respuestas usan códigos HTTP adecuados y mensajes de error claros para parámetros inválidos.		



Restricciones	
Sólo se mostrarán restaurantes aprobados y, por defecto, con al menos una reseña válida si así se configura (opcional).	

5. METODOLOGÍA

El desarrollo de **Foodie Rank** se basa en la metodología **Ágil SCRUM**, utilizando iteraciones cortas denominadas **Daily Sprints**, en las cuales se planifica, desarrolla y evalúa el progreso del producto cada día.

Roles

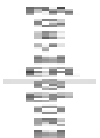
- **Product Owner:** *Fabián* – responsable de definir las funcionalidades y priorizar el backlog del producto.
- **Scrum Master:** *Carlos Mario* – encargado de facilitar el proceso Scrum, eliminar obstáculos y asegurar la comunicación fluida.
- **Developers:** *Fabián y Carlos Mario* – responsables de diseñar, desarrollar y probar las funcionalidades de la aplicación.

Herramientas

- **Repositorio:** GitHub (uso de ramas, commits convencionales y control de versiones).
- **Tablero Scrum:** Trello o GitHub Projects.
- **Comunicación:** Reuniones diarias (Daily Standup) y revisiones al final de cada sprint.

Ciclo de Trabajo Diario

1. **Sprint Planning:** Definir tareas y prioridades del día.
2. **Daily Stand Up:** Revisión del progreso y obstáculos.
3. **Development:** Implementación de código backend y frontend.
4. **Testing & Review:** Pruebas funcionales y push al repositorio.
5. **Sprint Retrospective:** Evaluación del día y mejora continua.



6. EVIDENCIA DE PLANTEAMIENTO DE PLATAFORMA DE TRABAJO

Acá se debe documentar toda la evidencia de trabajo colaborativo con los siguientes elementos:

- Link del repositorio donde se evidencia el trabajo colaborativo con el correcto uso de roles, ramas y conventional commit.
- Link de los videos que se grabaron en las reuniones realizadas (Sprint planning, Daily Stand Up, Sprint Retrospective)
- Evidencia (capturas) del tablero Scrum utilizado en todas las etapas donde se visualicen todas las tareas, requerimientos e historias de usuario documentadas allí con responsables, tiempos, priorización, y demás requerimientos que se indiquen para las tareas.
- Documentación de los resultados y funcionamiento del producto final resaltando tecnologías utilizadas y cumplimiento de trabajo basado en las tareas planteadas.

7. CONCLUSIONES

Conclusiones generales del documento:

El desarrollo de **Foodie Rank** permitió implementar buenas prácticas de desarrollo backend con Node.js y Express, reforzando la comprensión de JWT, validaciones, roles y estructura modular.

El trabajo colaborativo bajo la metodología **Scrum** facilitó la organización, priorización y cumplimiento de los objetivos diarios, logrando un producto funcional y escalable.

Conclusiones de la retrospectiva del Sprint:

- Se evidenció una comunicación efectiva entre los integrantes.
- Se identificó la necesidad de planificar mejor las tareas de validación y pruebas.
- Se alcanzaron los objetivos propuestos para el sprint diario con resultados satisfactorios.
- Se acordó mejorar la documentación y automatización de pruebas para próximos ciclos.