APLICACIÓN FULLSTACK RESTAURANTE TIANA'S

VALENTINA DELGADO RINCÓN
LAURA CAMILA FLOREZ SANTOS

GRUPO NOC

PEDRO GOMEZ

CAMPUSLANDS
SALON U1
RUTA NODE.JS
FLORIDABLANCA
2025

APLICACIÓN FULLSTACK RESTAURANTE TIANA'S

1. SITUACIÓN PROBLEMA

En la actualidad, muchos usuarios interesados en descubrir nuevos lugares para comer enfrentan dificultades al buscar información confiable y actualizada sobre restaurantes y platos. Las plataformas existentes suelen ofrecer datos incompletos, carecen de mecanismos de validación de opiniones o no permiten una interacción transparente entre los usuarios y los administradores de los establecimientos.

Por otro lado, los propietarios de restaurantes y administradores gastronómicos no cuentan con una herramienta centralizada que les permita gestionar reseñas, calificaciones, categorías y rankings de manera eficiente y segura. Esto genera diversos inconvenientes como:

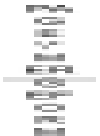
- **Dificultad para administrar información** de usuarios, reseñas y menús en un solo sistema.
- **Ausencia de control de acceso y permisos**, lo que puede provocar errores o pérdida de datos sensibles.
- **Problemas de seguridad**, derivados de sistemas que no implementan autenticación ni validaciones adecuadas.

Ante este panorama, surge la necesidad de desarrollar una **aplicación web** que permita a los usuarios **registrar, calificar y rankear restaurantes y platos**, mientras que los administradores puedan **gestionar usuarios, reseñas, categorías y rankings** con diferentes niveles de permisos y funciones.

Para garantizar la confiabilidad y seguridad del sistema, la aplicación debe contar con:

- **Autenticación segura** mediante JWT y cifrado de contraseñas.
- **Validaciones robustas** para todas las operaciones.
- **Control de abuso y manejo de errores centralizado.**
- **Persistencia transaccional en MongoDB**, asegurando la consistencia de los datos.
- **Frontend independiente**, desarrollado con HTML, CSS y JavaScript puro, que consuma la API del backend.

La implementación de este sistema permitirá centralizar la información, automatizar la gestión de reseñas y rankings y ofrecer una experiencia segura y escalable tanto para usuarios como para administradores. De esta forma, se busca optimizar la interacción entre clientes y restaurantes y promover la mejora continua de servicios.



2. LEVANTAMIENTO DE REQUERIMIENTOS

Fecha del levantamiento: 27/10/2025

Empresa solicitante: Tiana's - Restaurante

Método utilizado: Conferencia y entrevista

Con el fin de desarrollar una aplicación fullstack para la empresa Tiana's se llevó a cabo un proceso de levantamiento de requerimientos mediante dos métodos principales: una conferencia grupal con el equipo directivo y operativo del restaurante, y una entrevista individual con el gerente general y personal clave del área de operaciones y marketing.

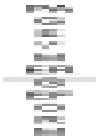
Durante estas sesiones, se recolectó información esencial para entender el funcionamiento actual de la empresa, sus necesidades tecnológicas, y los procesos que se desean automatizar. Y con esa información se llevaron a cabo los siguientes requerimientos:

3. REQUERIMIENTOS

3.1. Requerimientos Funcionales

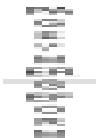
Categoría	Requerimiento
Requerimiento funcionales	**Gestión de usuarios:** Registro, inicio de sesión y autenticación mediante JWT. Roles: usuario y administrador. Los administradores pueden aprobar o rechazar restaurantes y platos.
Requerimientos funcionales	**Gestión de restaurantes:** CRUD de restaurantes. Solo los administradores pueden aprobar nuevas entradas. Validación para evitar nombres repetidos.
Requerimientos funcionales	**Gestión de platos:** CRUD de platos asociados a restaurantes. Validación para evitar duplicados. Atributos: nombre, descripción, categoría, ubicación, imagen opcional.

Requerimientos funcionales	**Gestión de reseñas:** Crear, editar y eliminar reseñas (usuarios autenticados). Cada reseña incluye comentario, calificación (1–5 estrellas) y fecha.
Requerimientos funcionales	**Likes y dislikes:** Los usuarios pueden dar “like” o “dislike” a reseñas de otros (no a las propias).
Requerimientos funcionales	**Rankings y listados:** Calcular y mostrar el ranking ponderado de restaurantes según calificaciones, likes/dislikes y fecha. Permitir ordenamiento y filtrado por categoría.
Requerimientos funcionales	**Gestión de categorías:** CRUD de categorías (solo administradores). Ejemplo: Comida rápida, Gourmet, Vegetariano, etc.
Requerimientos funcionales	**Documentación de API:** Generar documentación de los endpoints usando Swagger UI Express.
Requerimientos funcionales	**Frontend:** Interfaz en HTML, CSS y JS puro. Pantallas: inicio, login/registro, listado de restaurantes, detalle de restaurante, panel administrador. Consumo de API mediante fetch.
Requerimientos funcionales	**Planeación y control:** Uso de metodología SCRUM: definición de roles, sprints, historias de usuario y seguimiento en herramienta colaborativa (Trello, GitHub Projects o similar).



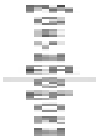
3.2. Requerimientos No Funcionales

Categoría	Requerimientos
Requerimientos No funcionales	**Seguridad:** Autenticación con JWT, manejo de contraseñas cifradas con bcrypt, protección de rutas según roles, uso de variables de entorno (.env) para credenciales.
Requerimientos No funcionales	**Rendimiento:** Implementación de express-rate-limit para limitar peticiones y prevenir abusos. Optimización de consultas en MongoDB.
Requerimientos No funcionales	**Disponibilidad y confiabilidad:** Manejo de errores centralizado. Uso de transacciones en MongoDB para operaciones críticas (crear reseñas o likes).
Requerimientos No funcionales	**Escalabilidad:** Arquitectura modular (models, controllers, routes, middlewares, services, config, utils) que permita extensión futura.
Requerimientos No funcionales	**Usabilidad:** Interfaz web intuitiva y responsive. Feedback visual (mensajes de error, validaciones).
Requerimientos No funcionales	**Mantenibilidad:** Código documentado, modular y versionado con control semántico (semver).
Requerimientos No funcionales	**Compatibilidad:** Conexión segura entre frontend y backend mediante CORS configurado.
Requerimientos No funcionales	**Documentación técnica:** Endpoints documentados con Swagger. README con instrucciones de instalación, configuración y ejemplos de uso.



3.3. Requerimientos Especiales

Categoría	Requerimientos
Requerimientos Especiales	**Tecnologías obligatorias:**Backend: Node.js + Express, dotenv, express-validator, express-rate-limit, mongodb (sin mongoose), passport-jwt, jsonwebtoken, bcrypt, swagger-ui-express.
Requerimientos Especiales	**Transacciones:** Uso obligatorio de transacciones en MongoDB para mantener consistencia de datos (por ejemplo, al registrar reseñas o modificar rankings).
Requerimientos Especiales	**Entrega del proyecto:** Repositorio backend y frontend separados. Video de demostración (máximo 10 min) mostrando el código y funcionamiento.
Requerimientos Especiales	**Gestión del desarrollo:** Implementación bajo metodología SCRUM, con documento de planeación en PDF adjunto al repositorio backend.
Requerimientos Especiales	**Seguridad adicional:** Validaciones robustas con express-validator. Protección de endpoints críticos con middlewares personalizados.



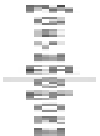
4. HISTORIAS DE USUARIO CON CRITERIOS DE ACEPTACIÓN

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF01	**Actor**	Usuario
NOMBRE DEL REQUERIMIENTO	**Registro, inicio de sesión y autenticación mediante JWT**		
Descripción			
Como usuario, puedo registrarme e iniciar sesión en el sistema de forma segura mediante JWT, para acceder a las funcionalidades de la aplicación según mi rol (usuario o administrador).			
Funcionalidad			
Implementar registro, inicio de sesión y autenticación con JWT y bcrypt, diferenciando permisos de usuario y administrador.			
Criterios de aceptación	1. El sistema debe permitir registro y autenticación con JWT. 2. El sistema validar correos únicos y contraseñas cifradas. 3. El sistema debe permitir a los administradores tener acceso a rutas exclusivas.		
Restricciones			
No se permite registro duplicado y el token debe incluir tiempo de expiración.			

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF02	**Actor**	Administrador
NOMBRE DEL REQUERIMIENTO	**CRUD de restaurantes con aprobación administrativa**		
Descripción			
Como administrador, puedo crear, editar, aprobar o eliminar restaurantes para mantener la calidad de los datos y evitar duplicados.			
Funcionalidad			
Desarrollar CRUD con validaciones y control de aprobación exclusivo de administradores.			
Criterios de aceptación	1. El sistema debe permitir solo a administradores aprobar o rechazar restaurantes. 2. El sistema debe permitir validar nombres únicos de restaurantes. 3. El sistema debe permitir registrar usuario que aprueba.		
Restricciones			
Operaciones limitadas a rol administrador.			

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF03	**Actor**	Administrador
NOMBRE DEL REQUERIMIENTO	**CRUD de platos asociados a restaurantes**		
Descripción			
Como administrador, puedo gestionar los platos de cada restaurante para mantener su información actualizada y evitar duplicados.			
Funcionalidad			
Permitir CRUD de platos asociados a un restaurante con validaciones de nombre, descripción y categoría.			
Criterios de aceptación	1. El sistema debe asociar platos a un restaurante existente. 2. El sistema validar nombres únicos por restaurante. 3. El sistema debe permitir guardar datos de manera persistente.		
Restricciones			
Solo administradores pueden crear o eliminar platos.			

HISTORIA DE USUARIO			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	RF04	**Actor**	Usuario
NOMBRE DEL REQUERIMIENTO	**Crear, editar y eliminar reseñas**		
Descripción			
Como usuario autenticado, puedo crear, modificar o eliminar reseñas sobre restaurantes o platos para compartir mi experiencia.			
Funcionalidad			
Implementar CRUD de reseñas con campos de comentario, calificación (1–5) y fecha.			
Criterios de aceptación	1. El sistema debe validar que solo usuarios autenticados puedan crear reseñas. 2. El sistema debe validar la calificación entre 1 y 5. 3. El sistema debe permitir editar o eliminar solo las reseñas propias.		
Restricciones			
Un usuario no puede reseñar el mismo restaurante dos veces.			



HISTORIA DE USUARIO			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	RF05	**Actor**	Usuario
NOMBRE DEL REQUERIMIENTO	**Votación de reseñas**		
Descripción			
Como usuario autenticado, puedo dar “like” o “dislike” a las reseñas de otros para valorar su utilidad.			
Funcionalidad			
Registrar votos positivos o negativos en reseñas de otros usuarios.			
Criterios de aceptación	1. El sistema debe validar que un usuario no vote por su propia reseña. 2. El sistema debe permitir un voto por reseña por usuario.		
Restricciones			
No se permiten votos duplicados.			

HISTORIA DE USUARIO			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	RF06	**Actor**	Usuario
NOMBRE DEL REQUERIMIENTO	**Listado y ranking de restaurantes**		
Descripción			
Como usuario, puedo visualizar un ranking de restaurantes basado en calificaciones, likes, dislikes y fecha de reseñas.			
Funcionalidad			
Calcular ranking ponderado y mostrar listado con filtros y ordenamientos.			
Criterios de aceptación	1. El sistema debe permitir ver el ranking actualizado automáticamente. 2. El sistema debe permitir filtrado por categoría. 3. El sistema debe permitir a los administradores tener acceso a rutas exclusivas.		
Restricciones			
Solo incluir restaurantes con al menos una reseña.			

HISTORIA DE USUARIO			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	RF07	**Actor**	Administrador
NOMBRE DEL REQUERIMIENTO	**CRUD de categorías**		
Descripción			
Como administrador, puedo crear, editar o eliminar categorías de comida para organizar los restaurantes.			
Funcionalidad			
Implementar CRUD de categorías con validaciones de nombres duplicados.			
Criterios de aceptación	1. El sistema debe validar la unicidad de nombres. 2. El sistema debe bloquear la eliminación si está en uso.		
Restricciones			
Solo administradores autenticados pueden acceder.			

HISTORIA DE USUARIO			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	RF08	**Actor**	Desarrollador
NOMBRE DEL REQUERIMIENTO	**Documentación con Swagger**		
Descripción			
Como desarrollador, puedo documentar los endpoints de la API usando Swagger UI Express para facilitar su uso.			
Funcionalidad			
Generar documentación interactiva con Swagger.			
Criterios de aceptación	1. Documentar todos los endpoints. 2. Incluir ejemplos de peticiones y respuestas.		
Restricciones			
Acceso disponible desde /api/docs.			

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF09	**Actor**	Usuario
NOMBRE DEL REQUERIMIENTO	**Interfaz HTML, CSS y JS**		
Descripción			
Como usuario, puedo interactuar con la aplicación mediante una interfaz web amigable y responsive.			

HISTORIA DE USUARIO			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	RF10	**Actor**	Equipo de desarrollo
NOMBRE DEL REQUERIMIENTO	**Planeación y control del proyecto**		
Descripción			
Como equipo de desarrollo, puedo organizar el proyecto bajo metodología SCRUM con roles, sprints y seguimiento.			
Funcionalidad			
Definir sprints, roles y tareas en GitHub Projects.			
Criterios de aceptación	1. Al menos 2 sprints definidos. 2. Documento PDF de planeación adjunto al repositorio.		
Restricciones			
Seguimiento obligatorio en herramienta colaborativa.			

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF11	**Actor**	Desarrollador backend
NOMBRE DEL REQUERIMIENTO	**Autenticación y cifrado seguro**		
Descripción			
Como desarrollador, puedo implementar autenticación con JWT y cifrado de contraseñas con bcrypt para proteger los datos del sistema.			
Funcionalidad			
Configurar autenticación segura, roles y manejo de variables .env.			
Criterios de aceptación	1. El sistema debe permitir usar bcrypt para contraseñas. 2. El sistema debe usar variables de entorno para credenciales. 3. El sistema debe proteger rutas sensibles.		
Restricciones			
No almacenar contraseñas en texto plano.			

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF12	**Actor**	Desarrollador
NOMBRE DEL REQUERIMIENTO	**Manejo de errores y transacciones**		
Descripción			
Como desarrollador, puedo manejar errores centralizados y usar transacciones en MongoDB para operaciones críticas.			
Funcionalidad			
Configurar manejo de errores global y transacciones.			
Criterios de aceptación	1. El sistema debe permitir implementar transacciones en creación de reseñas y likes. 2. El sistema debe registrar errores con mensajes claros.		
Restricciones			
Evitar pérdida de datos en fallos concurrentes.			

HISTORIA DE USUARIO			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	RF13	**Actor**	Desarrollador
NOMBRE DEL REQUERIMIENTO	**Arquitectura modular**		
Descripción			
Como desarrollador, puedo estructurar el sistema en módulos independientes para facilitar futuras ampliaciones.			
Funcionalidad			
Separar componentes en carpetas `/models`, `/controllers`, `/routes`, `/middlewares`, `/services`, `/config`, `/utils`.			
Criterios de aceptación	1. Cada módulo debe ser independiente. 2. Cumplir estándares de arquitectura limpia.		
Restricciones			
No se permite lógica de negocio en rutas.			

HISTORIA DE USUARIO			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	RF14	**Actor**	Usuario
NOMBRE DEL REQUERIMIENTO	**Interfaz intuitiva y responsive**		
Descripción			
Como usuario, puedo navegar fácilmente por la aplicación y recibir retroalimentación visual ante errores o validaciones.			
Funcionalidad			
Implementar interfaz responsive y mensajes dinámicos.			
Criterios de aceptación	1. El sistema debe permitir diseño responsive en pantallas principales. 2. El sistema debe mostrar alertas visuales ante errores.		
Restricciones			
Debe ser usable desde dispositivos móviles.			

HISTORIA DE USUARIO			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	RF15	**Actor**	Desarrollador
NOMBRE DEL REQUERIMIENTO	**Código documentado y versionado**		
Descripción			
Como desarrollador, puedo mantener el proyecto documentado y versionado bajo semver para facilitar futuras actualizaciones.			
Funcionalidad			
Documentar funciones, controladores y versiones.			
Criterios de aceptación	1. Seguir control semántico de versiones. 2. Comentarios en código principales.		
Restricciones			
No modificar código sin actualizar versión.			

HISTORIA DE USUARIO			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	RF16	**Actor**	Desarrollador frontend

NOMBRE DEL REQUERIMIENTO	**Comunicación segura entre frontend y backend**
Descripción	
Como desarrollador frontend, puedo conectar la interfaz al backend mediante CORS configurado correctamente.	
Funcionalidad	
Configurar CORS para permitir peticiones seguras desde el frontend.	
Criterios de aceptación	1. El sistema debe permitir acceso solo al dominio del frontend. 2. El sistema debe bloquear orígenes no autorizados.
Restricciones	
No exponer credenciales en el cliente.	

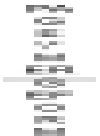
HISTORIA DE USUARIO			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	RF17	**Actor**	Desarrollador
NOMBRE DEL REQUERIMIENTO	**README y Swagger**		
Descripción			
Como desarrollador, puedo documentar el backend en un README con instalación, configuración y ejemplos de endpoints.			
Funcionalidad			
Crear README con estructura técnica del proyecto.			
Criterios de aceptación	1. El readme debe incluir pasos de instalación y variables .env. 2. Añadir ejemplos de endpoints.		
Restricciones			
Debe actualizarse con cada versión.			

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF18	**Actor**	Desarrollador
NOMBRE DEL REQUERIMIENTO	**Uso de dependencias y librerías especificadas**		
Descripción			
Como desarrollador, puedo implementar las tecnologías requeridas (Node.js, Express, MongoDB, JWT, bcrypt, dotenv, etc.) para cumplir con los lineamientos técnicos.			
Funcionalidad			
Integrar las dependencias obligatorias del proyecto.			

Criterios de aceptación	1. El sistema debe verificar que todas las librerías estén instaladas. 2. Documentar su uso en el README.
Restricciones	
No se permite el uso de frameworks no especificados.	

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF19	**Actor**	Desarrollador
NOMBRE DEL REQUERIMIENTO	**Uso de transacciones en MongoDB**		
Descripción			
Como desarrollador, puedo implementar transacciones en MongoDB para garantizar la consistencia de los datos.			
Funcionalidad			
Usar transacciones en operaciones críticas como reseñas o likes.			
Criterios de aceptación	1. Las operaciones deben ser atómicas. 2. El sistema debe registrar rollback en caso de error.		
Restricciones			
No se permiten operaciones parciales.			

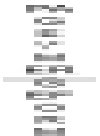
HISTORIA DE USUARIO			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	RF20	**Actor**	Equipo de desarrollo
NOMBRE DEL REQUERIMIENTO	**Repositorios y video de demostración**		
Descripción			
Como equipo, puedo entregar el proyecto en repositorios separados de backend y frontend con un video demostrativo.			
Funcionalidad			
Subir código a GitHub y grabar video funcional.			
Criterios de aceptación	1. Repositorios separados y públicos. 2. Video de máximo 10 minutos.		
Restricciones			
Todos los integrantes deben aparecer en el video.			



HISTORIA DE USUARIO			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	RF21	**Actor**	Equipo scrum
NOMBRE DEL REQUERIMIENTO	**Implementación de metodología SCRUM**		
Descripción			
Como equipo, puedo aplicar SCRUM para la gestión del desarrollo mediante roles, reuniones y backlog definidos.			
Funcionalidad			
Documentar roles, backlog y sprints en Trello o GitHub Projects.			
Criterios de aceptación	1. Definir Product Owner, Scrum Master y Developers. 2.Mostrar evidencias del seguimiento SCRUM.		
Restricciones			
Documento PDF obligatorio en el repositorio backend.			

HISTORIA DE USUARIO			
Prioridad: Alta			
CÓDIGO DEL REQUERIMIENTO:	RF22	**Actor**	Desarrollador backend
NOMBRE DEL REQUERIMIENTO	**Validaciones robustas y middlewares**		
Descripción			
Como desarrollador backend, puedo proteger endpoints con validaciones y middlewares personalizados para asegurar la integridad del sistema.			
Funcionalidad			
Implementar express-validator y middlewares de autorización.			
Criterios de aceptación	1. El sistema debe permitir validar datos en cada endpoint. 2. El sistema debe bloquear accesos no autorizados.		
Restricciones			
No exponer rutas críticas sin autenticación.			

HISTORIA DE USUARIO			
Prioridad: Media			
CÓDIGO DEL REQUERIMIENTO:	RF23	**Actor**	Desarrollador
NOMBRE DEL REQUERIMIENTO	**Configuración de variables de entorno**		
Descripción			
Como desarrollador, puedo configurar credenciales y secretos mediante variables de entorno para evitar exposición de datos sensibles.			



Funcionalidad	
Usar dotenv para manejar credenciales del sistema.	
Criterios de aceptación	1. Archivo .env configurado correctamente. 2. No exponer claves en el repositorio.
Restricciones	
El archivo .env debe excluirse con .gitignore.	

5. METODOLOGÍA

Metodología de Trabajo Utilizada: Kanban

Para el desarrollo del sistema integral de gestión solicitado por la empresa Tiana's, se ha optado por utilizar la metodología ágil Kanban, con el objetivo de facilitar una gestión visual y flexible del trabajo, permitiendo adaptarse rápidamente a cambios y priorizar tareas de forma eficiente.

Kanban es una metodología centrada en la entrega continua, la mejora constante del flujo de trabajo y la transparencia del proceso, lo que resulta ideal para proyectos donde los requerimientos pueden evolucionar durante el desarrollo.

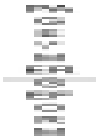
Estructura del Equipo y Roles Asignados

Para garantizar una organización efectiva y una comunicación fluida durante todo el desarrollo del proyecto, se han asignado los siguientes roles clave:

● **Product Owner: (Valentina Delgado)** Responsable de representar a la empresa Tiana's. Su función principal es priorizar las funcionalidades, definir los criterios de aceptación y asegurarse de que el producto final cumpla con las necesidades del cliente.

● **Scrum Master: (Camila Florez)** Encargado de facilitar el proceso de desarrollo, eliminar obstáculos, asegurar el cumplimiento de la metodología ágil y mantener al equipo enfocado y alineado con los objetivos del proyecto.

● **Equipo de Desarrolladores: (Camila Florez, Valentina Delgado)** Compuesto por los programadores encargados del diseño, desarrollo, prueba y mejora continua del sistema. Son responsables de transformar los requerimientos en funcionalidades reales, cumpliendo con los estándares técnicos y de calidad definidos.



Ventajas del Enfoque Kanban para Tiana's

- Mayor visibilidad del progreso del proyecto.
- Posibilidad de adaptarse rápidamente a cambios o nuevas solicitudes.
- Entregas parciales continuas, lo que permite validar funcionalidad paso a paso.

6. EVIDENCIA DE PLANTEAMIENTO DE PLATAFORMA DE TRABAJO

- Link **Repositorio GitHub Frontend**:

  https://github.com/ValentinaDelgadoRincon/restaurante_tiana-s-front

- Link **Repositorio GitHub Backend**:

  https://github.com/CamilaFlorez12/restaurante_tiana-s_backend

- Link video **Sprint Planning**:

  https://drive.google.com/file/d/1xf4ZLEhxDyYkV3sHKI9OW7O6Psik7dzc/view?usp=sharing

- Link video **Daily Stand Up**:

  https://drive.google.com/file/d/13wdY_uK3ofcumL_-vB3yAheVrARphMNY/view?usp=drive_link

- Link video **Sprint Review**:

  https://drive.google.com/file/d/1QvaECwTawwrEHKY4vEJCRS3tvndlwbIJ/view?usp=drive_link

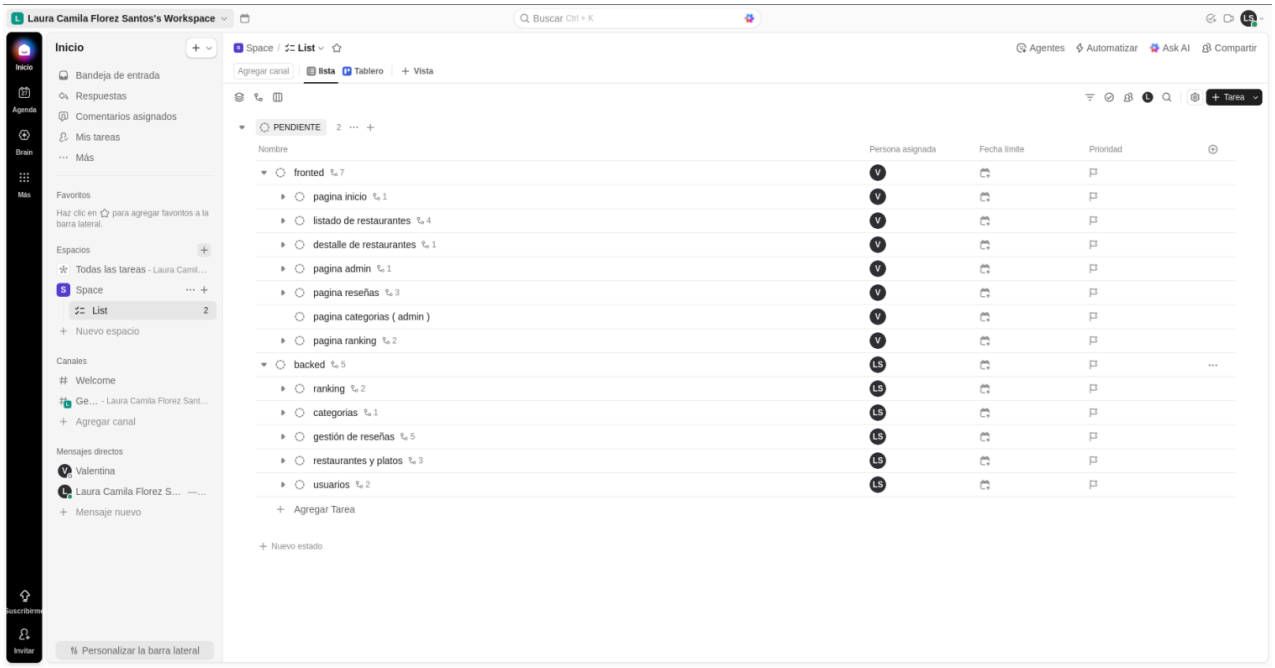
- Link video **Sprint Retrospective**:

  https://drive.google.com/file/d/1lfiq_NTHlNhI1bBAU9YRevlut-yAEjaB/view?usp=drive_link

● Link **Tablero Scrum**:

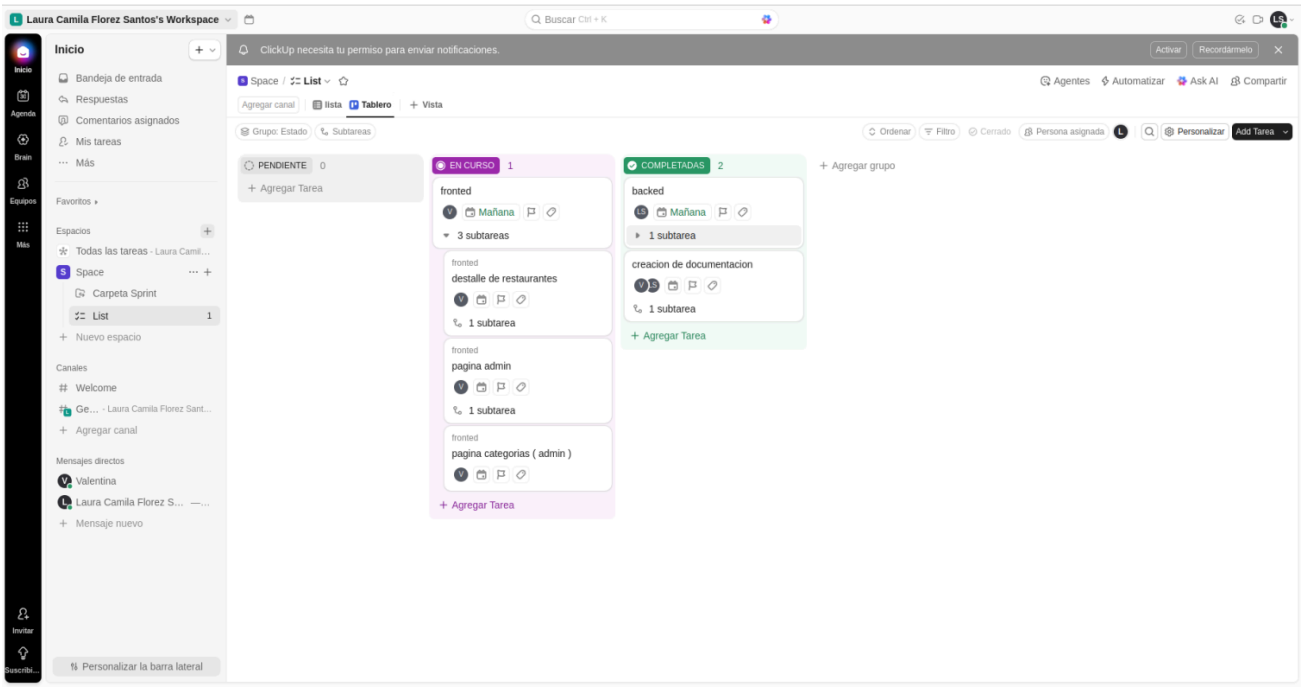
Backlog y asignación de tareas: En esta etapa se realizó la división y asignación de las tareas para el proyecto, a cada miembro del equipo se le asignó una tarea que a la vez estaba dividida en varias sub-tareas.

https://app.clickup.com/90132667992/v/b/li/901321736255



● Tareas en **Curso** y **Completadas**:

En esta etapa del proyecto los desarrolladores empezaron a realizar sus tareas y la actualización de estado de las Tareas es EN CURSO y algunas tareas sencillas se asignan como COMPLETADAS.



7. CONCLUSIONES

El proyecto “Tiana’s Reseñas” se desarrolla como una solución integral para modernizar la forma en que los usuarios califican y comentan sus experiencias gastronómicas. Actualmente, muchas reseñas se gestionan de forma dispersa en redes sociales o sitios sin control de autenticación, lo que dificulta el seguimiento y la confiabilidad de las opiniones. Este sistema centraliza la interacción entre usuarios, restaurantes y administradores, ofreciendo una experiencia completa, segura y dinámica.

Desde el punto de vista técnico, el sistema implementa una arquitectura MVC utilizando Node.js y Express en el backend, con controladores, servicios y rutas organizadas. La base de datos MongoDB almacena usuarios, restaurantes, platos y reseñas, garantizando persistencia, seguridad y escalabilidad. El frontend fue desarrollado con HTML, CSS y JavaScript modularizado, integrando el consumo de la API mediante fetch y autenticación con tokens JWT, lo que asegura una comunicación segura entre cliente y servidor.

En la gestión del proyecto, se aplicó la metodología Kanban, priorizando entregas continuas, revisión constante y comunicación efectiva. El equipo empleó herramientas como GitHub y ClickUp, logrando trazabilidad de cambios, organización del trabajo y coordinación ágil en cada sprint.

En conclusión, “Tiana’s Reseñas” constituye una base sólida tanto técnica como organizativa. Su diseño escalable y modular permitirá futuras mejoras, como integración de geolocalización, recomendaciones personalizadas y estadísticas avanzadas para administradores. El sistema optimiza la experiencia del usuario, fortalece la presencia digital de los restaurantes y garantiza una gestión eficiente de reseñas en línea.

Sprint Retrospective – Proyecto Tiana’s Reseñas

Duración del sprint: Del 29 al 31 de octubre de 2025

Equipo:

- **Product Owner:** Valentina Delgado
- **Scrum Master**: Laura Camila Flórez
- **Equipo de desarrollo**: Valentina Delgado, Laura Camila Flórez

Sprint objetivo: Construcción de la aplicación web completa para gestión de reseñas gastronómicas (usuarios, autenticación, restaurantes, reseñas y panel de administración).

1. Qué salió bien

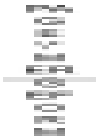
• Se logró la conexión exitosa entre el frontend y el backend mediante API REST, garantizando comunicación fluida con autenticación JWT.

• Se implementaron los módulos principales:

- Registro e inicio de sesión de usuarios y administradores.
- Listado dinámico de restaurantes con búsqueda y filtrado en tiempo real.
- Creación y visualización de reseñas asociadas a restaurantes y usuarios autenticados.
- Gestión administrativa de datos desde un panel con control de roles.

2. Qué se puede mejorar

• Algunos mensajes de error del servidor pueden ser más descriptivos para facilitar la depuración.



• Durante las pruebas iniciales se identificaron errores 401 y validaciones incompletas al enviar reseñas, lo cual se corrigió pero requiere reforzar el manejo de respuestas del backend.

• Se recomienda mejorar la gestión de tokens y sesiones, para permitir una expiración controlada y cierre de sesión automático.

• Se presentaron ligeros retrasos en el cronograma debido a la integración final entre los módulos de reseñas y autenticación.

3. Acciones para el próximo sprint

• Implementar validaciones más robustas en los formularios del frontend y respuestas más específicas desde el backend.

• Añadir un sistema de notificaciones o alertas visuales para mejorar la retroalimentación del usuario.

• Planificar mejor la distribución de tareas y establecer criterios de aceptación claros antes del desarrollo de cada módulo.