Obligatorio 1

Desarrollo Full Stack

Módulo backend

Octubre 2025

Santiago Neira Estudiante 334109

Cesar Martinez Estudiante 330903

Fecha de entrega 9/10/2025

# Descripción de la aplicación

# *API Cadetería sirve para gestionar envíos de paquetes de punta a punta: permite registrar clientes, crear pedidos con datos de origen y destino, consultar y filtrar envíos, seguir su estado (pendiente, en ruta, entregado, cancelado), reprogramar o actualizar información según permisos, y administrar categorías y usuarios. Está pensada para que empresas o particulares organicen y controlen sus envíos de forma simple y ordenada, con niveles de servicio diferenciados y un historial claro de cada pedido.*

# Entidades implementadas

# *• User: id, username, email, nombre, apellido, role (admin|cliente), plan (plus|premium), fechaCreacion, fechaActualizacion.*

# *• Envio: id, user, origen (Direccion), destino (Direccion), fechaRetiro, horaRetiroAprox, tamanoPaquete (chico|mediano|grande), notas, categoria (Categoria), estado (pendiente|en\_ruta|entregado|cancelado), codigoSeguimiento, fechas.*

# *• Direccion (VO): calle, numero, ciudad, referencia.*

# *• Categoria: nombre, descripcion.*

# *• Category (listado público): id, name.*

# URL productiva:

# *Despliegue Cesar:* [*apicadeteria-g62jum3rq-cesars-projects-2539e6a6.vercel.app*](https://apicadeteria-g62jum3rq-cesars-projects-2539e6a6.vercel.app/)

# *Despliegue Santiago:* [*https://apicadeteria-pmjgfv8nm-sanei1509s-projects.vercel.app*](https://apicadeteria-pmjgfv8nm-sanei1509s-projects.vercel.app/)

# Documentación de los endpoints

# *Cesar:* [*https://apicadeteria-g62jum3rq-cesars-projects-2539e6a6.vercel.app/docs*](https://apicadeteria-g62jum3rq-cesars-projects-2539e6a6.vercel.app/docs)

Santiago: [*https://apicadeteria-pmjgfv8nm-sanei1509s-projects.vercel.app/docs*](https://apicadeteria-pmjgfv8nm-sanei1509s-projects.vercel.app/docs)

# Reglas de negocio implementadas

Usuarios & autenticación

R1. Registro único: username/email no repetidos; datos obligatorios válidos; duplicado → 409.

R2. Login: credenciales válidas → emitir JWT; toda ruta bajo /v1/\* exige Bearer.

Planes

R3. Plan por defecto: al registrarse, el usuario queda en plus.

R4. Cambio de plan: de plus → premium (lo hace admin via /v1/users/plan); precondición: el usuario está en plus.

R5. Límite por plan: plus sólo puede dar de alta 10; premium sin límite.

Envíos

R6. Creación: requiere origen, destino, fechaRetiro (YYYY-MM-DD), tamanoPaquete (chico|mediano|grande); horaRetiroAprox HH:MM opcional; categoria.

R7. Listado y filtros: por estado, tamanoPaquete, fechas (fecha, fechaDesde/fechaHasta, ultimos=semana|mes).

R8. Actualización (cliente): no puede cambiar estado; puede reprogramar fechaRetiro hasta 1 día antes (no el mismo día).

R9. Actualización (admin): puede cambiar todos los datos de un envío, y es el único que puede cambiar el estado (pendiente|en\_ruta|entregado|cancelado).

R10. Eliminación (cliente): no se puede borrar si fechaRetiro es hoy o ya pasó.

Categorías

R11. Consulta pública de categorías.

Transversales

R12. Responder con códigos estándar coherentes (200/201/204/400/401/403/404/409/500) y mensajes consistentes según el caso.

# Colección de Postman:

*Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.*

# Usuarios de prueba

*Usuario plan plus:*

* *usuario: sanei1509*
* *contraseña: 1234*
* ***1 envío*** *(1 pendiente)*

*Usuario plan plus:*

* *usuario: usuario1*
* *contraseña: 1234*
* ***9 envíos*** *(9 pendientes)*

*Usuario premium:*

* *usuario: cesarito*
* *contraseña: 1234*
* ***12 envíos*** *(12 pendientes)*

*Usuario administrador:*

* *usuario: cesaritoAdmin (responde a mayúsculas)*
* *contraseña: 1234*

*Usuario administrador:*

* *usuario: adminS*
* *contraseña: 1234*

# Uso de IAG

Interpretación de requisitos y modelado

La IA ayudó a traducir el enunciado del obligatorio en endpoints concretos (públicos/privados), reglas de negocio (plan plus, reprogramación, eliminación) y esquemas de datos (Joi/Mongoose).

Propuso una estructura de carpetas y el orden correcto de middlewares (Swagger antes de rutas en Vercel, CORS y express.json()).

Diseño de validaciones

Generó el schema Joi con mensajes específicos (fecha ≥ hoy, formato HH:mm, límites de longitud).

Sugirió .unknown(true) y allowUnknown: true durante la validación para aislar problemas de compatibilidad y despliegue.

Robustez en búsqueda de categorías

Reescribió la lógica de lookup de categorías para aceptar categoryId o nombre, tolerante a mayúsculas/acentos, usando collation y regex de respaldo.

Aseguró que solo se acepten nombres existentes en DB (si no, 400).

DTO y consistencia de respuesta

Unificó la salida con categoria: { id, nombre } usando populate, independientemente de cómo vino la categoría en el request (id, nombre u objeto).

Debugging guiado por logs

Indicó logs quirúrgicos (ruta del controller, ruta del schema, body crudo y normalizado) que permitieron ubicar algunos problemas y confirmar que el controller/schema correctos estaban cargando en Vercel.

Casos de prueba reproducibles

Proporcionó un set de cURL positivos/negativos y un seed script opcional para poblar categorías, acelerando la verificación funcional.

Claridad documental (Swagger)

Ayudó a alinear OpenAPI con el comportamiento real (parámetros, ejemplos de body, notas sobre el wrap de loginReal, seguridad bearerAuth).

# Extras

*Documentación OpenAPI + Swagger UI*

***Health check (GET /)****: endpoint simple para verificar disponibilidad y monitoreo sin tocar rutas de negocio.*

***Reglas adicionales por rol****: el cliente no puede cambiar estado, sólo reprograma con ≥1 día de anticipación y no puede eliminar en fecha del retiro.*

***Filtros avanzados en listados****: por estado, tamaño, fecha puntual, rangos y “última semana/mes” para consultas más potentes.*