

Caso de uso especificacione

UC-01 Registrar ingreso de vehículo (genera OT)

Actores: Recepcionista, Guardia de Acceso (colabora)

Precondiciones: Usuario autenticado; vehículo llega al taller.

Flujo principal:

Recepcionista ingresa patente y datos del chofer.

Sistema valida que no exista ingreso activo para esa patente (RF-04).

Sistema crea OT con estado “Ingresado” y hora de ingreso (RF-02).

Opcional: adjunta fotos/checklist del ingreso (RF-03).

Sistema registra auditoría (RF-23).

Flujos alternos:

2a. Si hay ingreso activo → muestra error y sugiere búsqueda (RF-04).

Postcondiciones: OT creada, datos guardados y trazados.

Referencias: RF-01..RF-04, RF-23; RNF-05/07/12.

UC-02 Asignar OT a mecánico

Actor: Jefe de Taller

Pre: OT en estado “Ingresado” o “Diagnóstico”.

Flujo: selecciona OT → asigna mecánico → sistema registra evento y deja traza (RF-05, RF-07, RF-23).

Post: OT con responsable asignado; histórico actualizado.

Refs: RF-05/07/23; RNF-12.

UC-03 Actualizar estado de OT

Actor: Mecánico

Pre: OT asignada al mecánico.

Flujo: cambia estado (Diagnóstico/En reparación/En pausa/Finalizada) → valida transición → registra en historial (RF-06/RF-07).

Alternos: transición inválida → mensaje y no persiste.

Post: historial consistente; si “Finalizada”, notifica (ver UC-09).

Refs: RF-06/07/20/23; RNF-05/12.

UC-04 Registrar pausa de OT

Actor: Mecánico

Pre: OT en ejecución.

Flujo: selecciona motivo (catálogo) y comienza pausa → al reanudar, sistema calcula tiempo y deja traza (RF-06/07).

Refs: RF-06/07; RNF-12.

UC-05 Gestionar repuestos (consultar stock y entregar)

Actores: Asistente de Repuestos, API Inventario

Pre: OT activa.

Flujo: busca pieza → consulta API Inventario (RF-11) → si hay stock, registra entrega y actualiza inventario (RF-12) → traza (RF-23).

Alternos: caída API → registra incidencia (RF-13) y permite solicitud diferida.

Post: movimiento de repuesto asociado a OT.

Refs: RF-10..RF-14, RF-23; RNF-26..29.

UC-06 Cerrar OT

Actor: Jefe de Taller

Pre: OT “Finalizada” por mecánico.

Flujo: valida checklist y documentos → cierra OT (RF-08) → registra en historial (RF-07) → dispara notificación (UC-09).

Refs: RF-07/08/20/23.

UC-07 Registrar entrada/salida de vehículo

Actor: Guardia de Acceso

Pre: Vehículo en portería.

Flujo: registra entrada o salida con timestamp y evidencia (fotos) (RF-03) → traza (RF-23).

Refs: RF-03/23.

UC-08 Generar reportes y exportar

Actores: Jefe de Taller, Supervisor Zonal

Pre: Datos existentes.

Flujo: filtra por fechas/estado/mecánico → el sistema genera KPI y permite exportar PDF/XLS (RF-15..RF-19).

Rendimiento: p95 ≤ 5 s (RNF-07).

Refs: RF-15..RF-19; RNF-07/18.

UC-09 Enviar notificaciones por eventos

Actores: Sistema (timer/evento), API Notificaciones

Disparadores: creación/ cierre de OT, pausas prolongadas, vehículo listo.

Flujo: genera payload → envía vía API Notificaciones → registra log/estado de entrega (RF-20, RF-23).

Alternos: reintentos ante error (RNF-28); latencia p95 ≤ 3–5 s (RNF-29).

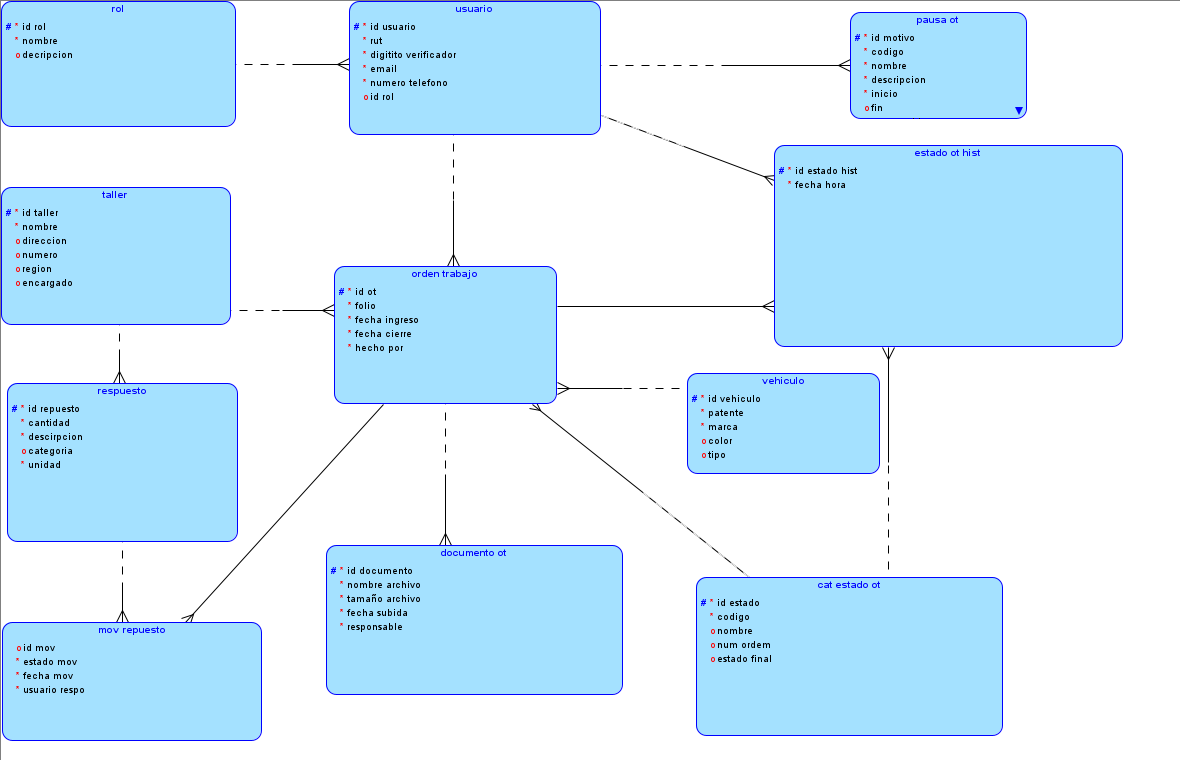
Refs: RF-20/23; RNF-26..29.

UC-10 Autenticación y gestión de usuarios/roles

Actores: Usuario, Administrador

Flujo: login con credenciales → asignación de permisos por rol → auditoría (RF-21..RF-25).

Refs: RF-21..RF-25; RNF-09..12/21..22.



MER

El modelo de base de datos propuesto para la *Plataforma de Gestión Digital de Ingreso de Vehículos al Taller* de PepsiCo Chile se estructura bajo una arquitectura relacional normalizada en Tercera Forma Normal (3FN), garantizando la eliminación de redundancias y la integridad de los datos. A continuación, se describen las entidades principales y sus relaciones de forma narrativa.

El usuario representa a toda persona que interactúa con el sistema (recepcionista, mecánico, jefe de taller, asistente de repuestos, supervisor, coordinador o administrador). Cada registro contiene su identificador, rut, nombres, apellidos, correo electrónico, contraseña cifrada, estado y último inicio de sesión. El usuario se vincula con la entidad rol, que define el perfil y los permisos asociados a cada tipo de actor dentro de la plataforma, permitiendo controlar las operaciones que puede realizar.

La entidad taller almacena la información de cada centro de mantenimiento donde se gestionan las órdenes de trabajo. Incluye datos como nombre, región y dirección física. Por su parte, la entidad vehículo conserva la información básica de los camiones o automóviles de la flota: patente, marca, modelo, año y número VIN. Un vehículo puede generar múltiples órdenes de trabajo a lo largo del tiempo, lo que se representa mediante una relación uno a muchos entre *vehículo* y *orden\_trabajo*.

La orden de trabajo (OT) es la entidad central del modelo y representa el proceso de ingreso, diagnóstico, reparación y cierre de un vehículo. Cada OT registra su folio único, las fechas de ingreso y cierre, el estado actual, observaciones, el usuario que la creó y el mecánico asignado. Se relaciona con las entidades *vehículo*, *taller* y *usuario* para mantener trazabilidad completa sobre quién la creó, dónde se atiende y a qué vehículo corresponde.

El historial de cambios de estado de cada orden se gestiona mediante la entidad estado\_ot\_hist, donde se almacena cada transición junto con la fecha, el usuario responsable y el tipo de estado, referenciado desde el catálogo cat\_estado\_ot, que define las fases válidas del proceso (por ejemplo: Ingresado, Diagnóstico, En reparación, En pausa, Finalizada, Cerrada). Este diseño permite mantener una auditoría detallada de la evolución de cada OT.

La entidad pausa\_ot registra las interrupciones del proceso productivo, especificando el motivo, las marcas de tiempo de inicio y fin, la duración total y el usuario responsable. Los motivos se administran en el catálogo cat\_motivo\_pausa, donde se clasifican las causas y se indica si son o no productivas, favoreciendo el análisis de eficiencia operativa.

Los documentos digitales asociados a las órdenes, tales como fotografías, checklists o informes técnicos, se almacenan en la entidad documento\_ot. Esta contiene el tipo de documento, la ruta de almacenamiento, el tamaño del archivo, el usuario que lo subió y la fecha de registro. Los tipos de documentos válidos se definen en el catálogo cat\_tipo\_doc, asegurando consistencia en la clasificación de la evidencia visual o documental.

El manejo de insumos se modela a través de las entidades repuesto y movimiento\_repuesto. La primera guarda el detalle de cada ítem disponible en bodega, incluyendo su código SKU, descripción y unidad de medida. La segunda registra los movimientos asociados a una OT —ya sea entrega o devolución— indicando la cantidad, la fecha y el responsable del movimiento. Este enfoque permite mantener la trazabilidad del uso de repuestos y su impacto en cada reparación.

Finalmente, todos los catálogos del sistema (cat\_estado\_ot, cat\_motivo\_pausa y cat\_tipo\_doc) funcionan como tablas maestras que evitan la repetición de valores y estandarizan la información utilizada por las tablas transaccionales. El modelo en su conjunto garantiza integridad referencial mediante claves foráneas y elimina dependencias transitivas, cumpliendo con las normas de normalización establecidas para bases de datos relacionales.

# 1) Explicación

# 

# – Modelo relacional

El modelo lógico organiza la información del sistema en tablas normalizadas que garantizan integridad y trazabilidad del proceso completo de la Orden de Trabajo (OT). Las entidades son: vehiculo, taller, usuario/rol y orden\_trabajo.

Desde la OT se desprenden los registros transaccionales que dan evidencia del ciclo de vida: estado\_ot\_hist (historial de estados), pausa\_ot (pausas con motivo), documento\_ot (evidencia digital) y movimiento\_repuesto (consumo/devoluciones de ítems). Para asegurar consistencia, se emplean catálogos (cat\_estado\_ot, cat\_motivo\_pausa, cat\_tipo\_doc) que estandarizan valores y facilitan auditoría.

dominante es 1:N: un *vehículo* genera muchas OTs, un *taller* atiende muchas OTs, y una OT tiene múltiples estados, pausas, documentos y movimientos de repuesto. Las claves foráneas materializan estas dependencias y evitan huérfanos; las columnas únicas (patente, folio, sku) impiden duplicados. El modelo favorece reportabilidad: consultas por fechas, estado, mecánico, taller o vehículo se optimizan con índices propuestos.

# **diccionario de Datos – Modelo Lógico (tablas principales)**

### 

rol

* id\_rol (uuid, N, PK): identificador del rol.
* nombre (varchar(30), N): ADMIN, JEFE\_TALLER, etc.
* descripcion (varchar(120), S): detalle.

usuario

* id\_usuario (uuid, N, PK)
* rut (varchar(12), N, ÚNICO)
* nombres (varchar(80), N)
* apellidos (varchar(80), N)
* email (varchar(120), N, ÚNICO)
* hash\_password (varchar(255), N)
* estado (varchar(15), N): ACTIVO/BLOQUEADO
* ultimo\_login (timestamp, S)
* id\_rol (uuid, N, FK→rol)

taller

* id\_taller (uuid, N, PK)
* nombre (varchar(60), N, ÚNICO)
* region (varchar(30), N)
* direccion (varchar(120), S)

vehiculo

* id\_vehiculo (uuid, N, PK)
* patente (varchar(10), N, ÚNICO)
* marca (varchar(40), S)
* modelo (varchar(40), S)
* anio (smallint, S)
* vin (varchar(20), S, ÚNICO)

### 2.2 Catálogos

cat\_estado\_ot

* id\_estado (uuid, N, PK)
* codigo (varchar(10), N, ÚNICO): ING, DIAG, ENR, EPAU, FIN, CERR
* nombre (varchar(40), N)
* orden (smallint, N)
* es\_final (boolean, N)

cat\_motivo\_pausa

* id\_motivo (uuid, N, PK)
* codigo (varchar(20), N, ÚNICO)
* nombre (varchar(60), N)
* es\_productiva (boolean, N)

cat\_tipo\_doc

* id\_tipo\_doc (uuid, N, PK)
* codigo (varchar(20), N, ÚNICO): FOTO, CHECKLIST, INFORME
* nombre (varchar(60), N)

### 2.3 Transaccionales

orden\_trabajo

* id\_ot (uuid, N, PK)
* folio (varchar(20), N, ÚNICO)
* id\_vehiculo (uuid, N, FK→vehiculo)
* id\_taller (uuid, N, FK→taller)
* fecha\_ingreso (timestamp, N)
* fecha\_cierre (timestamp, S)
* id\_estado\_actual (uuid, N, FK→cat\_estado\_ot)
* observaciones (varchar(500), S)
* id\_usuario\_crea (uuid, N, FK→usuario)
* id\_usuario\_asignado (uuid, S, FK→usuario)

estado\_ot\_hist

* id\_estado\_hist (uuid, N, PK)
* id\_ot (uuid, N, FK→orden\_trabajo)
* fecha\_hora (timestamp, N)
* id\_estado (uuid, N, FK→cat\_estado\_ot)
* usuario\_resp (uuid, N, FK→usuario)
* comentario (varchar(300), S)

pausa\_ot

* id\_pausa (uuid, N, PK)
* id\_ot (uuid, N, FK→orden\_trabajo)
* id\_motivo (uuid, N, FK→cat\_motivo\_pausa)
* inicio (timestamp, N)
* fin (timestamp, S)
* duracion\_min (int, S)
* usuario\_resp (uuid, N, FK→usuario)

documento\_ot

* id\_doc (uuid, N, PK)
* id\_ot (uuid, N, FK→orden\_trabajo)
* id\_tipo\_doc (uuid, N, FK→cat\_tipo\_doc)
* nombre\_archivo (varchar(120), N)
* ruta (varchar(255), N)
* peso\_mb (numeric(6,2), N)
* subido\_por (uuid, N, FK→usuario)
* fecha\_subida (timestamp, N)

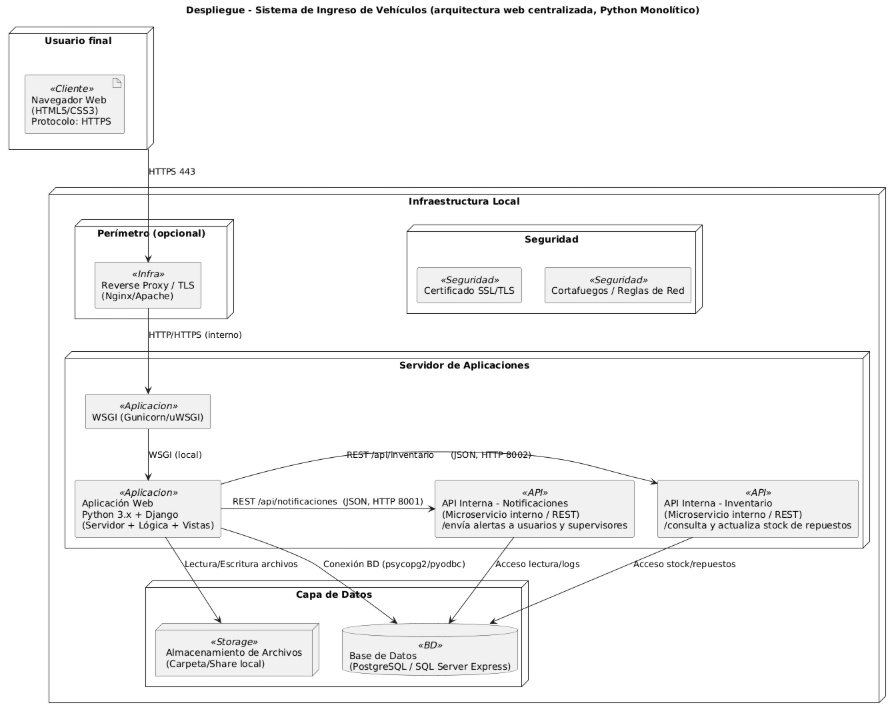
repuesto

* id\_repuesto (uuid, N, PK)
* sku (varchar(30), N, ÚNICO)
* descripcion (varchar(120), N)
* unidad (varchar(10), N)

movimiento\_repuesto

* id\_mov (uuid, N, PK)
* id\_ot (uuid, N, FK→orden\_trabajo)
* id\_repuesto (uuid, N, FK→repuesto)
* tipo (varchar(12), N, CHECK ∈ {ENTREGA, DEVOLUCION})
* cantidad (numeric(10,2), N, >0)
* fecha\_hora (timestamp, N)
* usuario\_resp (uuid, N, FK→usuario)
* obs (varchar(200), S)

uml despliegue



Este diagrama muestra cómo está distribuido el sistema físicamente y cómo se comunican sus partes durante el funcionamiento normal.

En la parte superior está el usuario, que accede al sistema desde un navegador web usando una conexión segura HTTPS.

Esa solicitud llega primero a un servidor perimetral o proxy, que se encarga de manejar la seguridad y reenviar las peticiones al servidor de aplicaciones, donde está instalada la aplicación web desarrollada en Python con Django.

Dentro de ese mismo servidor, además de la aplicación principal, existen dos APIs internas:

* una de notificaciones, que envía alertas o mensajes a los usuarios,
* y otra de inventario, que controla el stock de repuestos en tiempo real.

Cuando el sistema necesita guardar o consultar datos, se conecta a la base de datos relacional:  
 en el ambiente de desarrollo usa SQLite, y en producción se conecta a Oracle.  
 Además, las fotos y documentos que se suben quedan guardados en el repositorio local de archivos del servidor.

Finalmente, toda la infraestructura está protegida por certificados SSL/TLS y firewalls, que aseguran que la información viaje cifrada y que solo los usuarios autorizados puedan acceder.

En resumen, este diagrama muestra cómo cada parte del sistema colabora: el navegador envía las peticiones, el servidor Django las procesa, las APIs complementan funciones, la base de datos guarda la información y todo funciona dentro de un entorno seguro y controlado.”

@startuml

' ============ Estilo ============

skinparam componentStyle rectangle

skinparam shadowing false

hide circle

title Despliegue – Sistema de Ingreso de Vehículos (arquitectura web centralizada, Python Monolítico)

' ============ Usuario ============

node "Usuario final" as usr {

artifact "Navegador Web\n(HTML5/CSS3)\nProtocolo: HTTPS" as browser <<Cliente>>

}

' ============ Infraestructura ============

node "Infraestructura Local" as infra {

' (Opcional) Perímetro / Reverse Proxy

node "Perímetro (opcional)" as perimetro {

component "Reverse Proxy / TLS\n(Nginx/Apache)" as rp <<Infra>>

}

' Servidor de Aplicaciones (monolito web)

node "Servidor de Aplicaciones" as appsrv {

component "Aplicación Web\nPython 3.x + Django\n(Servidor + Lógica + Vistas)" as app <<Aplicacion>>

component "WSGI (Gunicorn/uWSGI)" as wsgi <<Aplicacion>>

' ===== APIs Internas =====

component "API Interna – Notificaciones\n(Microservicio interno / REST)\n/envía alertas a usuarios y supervisores" as apiNotif <<API>>

component "API Interna – Inventario\n(Microservicio interno / REST)\n/consulta y actualiza stock de repuestos" as apiInv <<API>>

}

' Capa de Datos

node "Capa de Datos" as data {

database "Base de Datos\n(PostgreSQL / SQL Server Express)" as db <<BD>>

node "Almacenamiento de Archivos\n(Carpeta/Share local)" as fs <<Storage>>

}

' Seguridad básica

node "Seguridad" as sec {

component "Certificado SSL/TLS" as tls <<Seguridad>>

component "Cortafuegos / Reglas de Red" as fw <<Seguridad>>

}

}

' ============ Conexiones ============

usr -down-> rp : HTTPS 443

rp -down-> wsgi : HTTP/HTTPS (interno)

wsgi -down-> app : WSGI (local)

app -down-> db : Conexión BD (psycopg2/pyodbc)

app -down-> fs : Lectura/Escritura archivos

' ===== Conexiones de APIs Internas =====

app -right-> apiNotif : REST /api/notificaciones (JSON, HTTP 8001)

app -right-> apiInv : REST /api/inventario (JSON, HTTP 8002)

apiNotif -down-> db : Acceso lectura/logs

apiInv -down-> db : Acceso stock/repuestos

' Si NO usas reverse proxy, descomenta esta línea y comenta la conexión a rp:

' usr -down-> wsgi : HTTPS 443

' ============ Leyenda ============

legend right

== Estereotipos ==

<<Cliente>> Navegador del usuario

<<Aplicacion>> Servidor de aplicación web (monolito Django)

<<API>> Servicios internos REST (Notificaciones / Inventario)

<<BD>> Base de datos relacional

<<Storage>> Archivos (fotos/documentos)

<<Infra>> Componente perimetral (opcional)

<<Seguridad>> Elementos de seguridad

== Puertos ==

443 HTTPS | 8000 WSGI | 8001/8002 APIs internas

endlegend

@enduml