# Arquitectura del Sistema

## 1. Descripción General

El presente diagrama representa la arquitectura física y lógica del sistema “Gestión de Ingreso de Vehículos PepsiCo”, mostrando cómo se distribuyen los componentes de software dentro de la infraestructura del servidor.  
 Su propósito es ilustrar la interacción entre los elementos que componen la solución, sus dependencias y el flujo de comunicación entre capas durante la operación normal del sistema.

El sistema se implementa bajo una arquitectura web centralizada, donde todos los servicios y recursos se concentran en un mismo entorno controlado, garantizando estabilidad, seguridad y eficiencia en la gestión de los procesos de taller.

## 2. Modelo Arquitectónico

El sistema se estructura según una arquitectura en tres capas, desarrollada bajo el framework Django en el lenguaje Python 3…., complementado con servicios internos que amplían la funcionalidad principal.

* Capa de Presentación:  
   Interfaz gráfica accesible desde un navegador web (HTML5, CSS3, JavaScript, Python).  
   Permite a los distintos roles —recepcionistas, mecánicos, supervisores y administradores— interactuar con el sistema.
* Capa de Aplicación:  
   Desarrollada en Django, operando bajo un servidor WSGI (Gunicorn/uWSGI).  
   Gestiona la lógica del negocio, incluyendo la creación y seguimiento de órdenes de trabajo, estados, gestión de repuestos, reportes y control de usuarios.  
   Esta capa también incorpora APIs internas:  
  + API de Notificaciones: encargada de enviar alertas y actualizaciones automáticas a los usuarios.
  + API de Inventario: que consulta y actualiza el stock de repuestos en tiempo real.
* Capa de Datos:  
   Responsable del almacenamiento y persistencia de información.  
   En el entorno de desarrollo utiliza SQLite (archivo local /app/db.sqlite3), mientras que en producción opera sobre Oracle 19c, garantizando confiabilidad y rendimiento.  
   Adicionalmente, los documentos e imágenes se guardan en un repositorio local de archivos (carpetas *static* y *media*).

Este modelo sigue el patrón arquitectónico en capas, donde cada nivel cumple un rol definido y se comunica únicamente con las capas adyacentes.  
 La estructura favorece la separación lógica, mejora la escalabilidad, facilita el mantenimiento y refuerza la seguridad del sistema.

| Capa | Componente | Tecnología / Lenguaje | Función principal |
| --- | --- | --- | --- |
| Presentación | Navegador Web | HTML5, CSS3, JavaScript,Python | Interfaz utilizada por los usuarios para registrar y consultar información del taller. |
| Aplicación | Servidor de Aplicaciones | Python 3.x con Django + WSGI (Gunicorn/uWSGI) | Ejecuta la lógica de negocio, controladores y generación dinámica de vistas. |
| Aplicación | API Interna – Notificaciones | Python + REST (HTTP 8001) | Envía alertas automáticas y registra logs de comunicación. |
| Aplicación | API Interna – Inventario | Python + REST (HTTP 8002) | Consulta y actualiza stock de repuestos en el sistema. |
| Datos | Base de Datos Relacional | SQLite (dev) / Oracle 19c (prod) | Almacena información estructurada: usuarios, vehículos, órdenes de trabajo, repuestos, estados y reportes. |
| Datos | Almacenamiento de Archivos | Repositorio local (*static/media*) | Conserva documentos e imágenes asociadas a las órdenes de trabajo. |
| Seguridad | Capa de Conexión Segura | HTTPS con Certificado SSL/TLS + Firewall | Protege la comunicación y restringe el acceso no autorizado. |

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 4. Descripción del Diagrama de Despliegue

El Diagrama UML de Despliegue ilustra la organización física de los componentes dentro de la infraestructura tecnológica:

* Los usuarios acceden al sistema desde un navegador web, utilizando una conexión segura HTTPS hacia el servidor perimetral (Nginx/Apache).
* Las solicitudes son derivadas al servidor de aplicaciones Python, donde se ejecuta Django sobre WSGI.
* En este servidor se encuentran también las APIs internas de Notificaciones e Inventario, que funcionan como microservicios REST locales y se comunican mediante puertos internos (8001 y 8002).
* El servidor de aplicaciones interactúa con la base de datos relacional, consultando o actualizando información según las operaciones solicitadas.  
  Los documentos e imágenes cargados por los usuarios se almacenan en el repositorio local de archivos.
* Todo el entorno se asegura con un certificado SSL/TLS y un cortafuegos que controla las conexiones entrantes y salientes

## 5. Flujo General de Comunicación

* El usuario accede al sistema desde su navegador mediante una conexión HTTPS (443).
* El reverse proxy recibe la petición y la redirige al servidor WSGI, donde se ejecuta Django.
* Django procesa la solicitud, ejecuta la lógica de negocio y, cuando corresponde:  
  + Invoca la API de Notificaciones para enviar alertas.
  + O consulta la API de Inventario para validar o registrar movimientos de repuestos.
* El servidor accede a la base de datos relacional (SQLite en DEV / Oracle en PROD) y al repositorio de archivos para recuperar o almacenar datos.
* El resultado se devuelve al navegador del usuario en forma de interfaz actualizada (HTML dinámico).

Todo el flujo se mantiene cifrado bajo TLS 1.2+, asegurando la integridad y confidencialidad de la información.

## 6. Conclusión

El sistema implementa una arquitectura web monolítica y centralizada, desarrollada en Python con Django, que concentra la interfaz, la lógica de negocio y la gestión de datos dentro de un entorno controlado.  
 Este enfoque simplifica la administración y despliegue de la aplicación, garantizando seguridad, estabilidad y mantenibilidad.

La incorporación de APIs internas permite extender las capacidades del sistema sin aumentar el acoplamiento, facilitando la integración de nuevas funciones (por ejemplo, control de inventario o alertas automáticas).  
 Gracias a su arquitectura en capas, el sistema puede evolucionar progresivamente, integrarse con otros servicios corporativos de PepsiCo y mantener la consistencia de los datos sin comprometer la operación principal del taller.