ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

ESPOCH

SABER PARA SER

Nombre:

Alex Vallejo

Código:

6912

Materia:

Aplicaciones Informáticas II

Fecha:

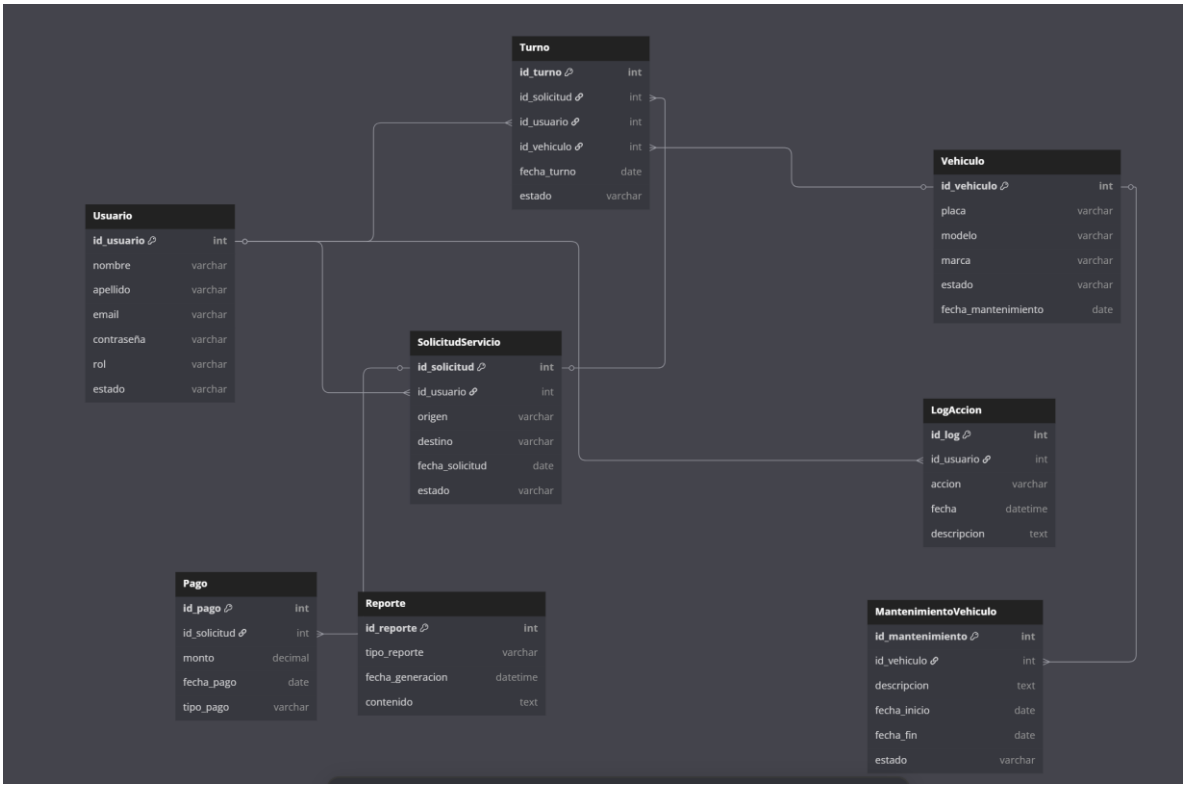
16/05/2025

Base de datos

Diagrama Entidad-Relación (ER)

Un Diagrama Entidad-Relación (ER) es una representación gráfica que muestra la estructura lógica de una base de datos. Utiliza símbolos para describir las entidades (objetos o conceptos relevantes), sus atributos (propiedades o características), y las relaciones entre estas entidades. El diagrama ER es parte del proceso para diseñar una base de datos normalizada. La normalización es el proceso de organizar los datos para minimizar la redundancia y dependencias indeseadas. El diseño normalizado facilita la eficiencia en las consultas y mantenimiento del sistema.

El diseño lógico (representado por el diagrama ER) se transforma luego en el diseño físico (tablas, índices, constraints) que será implementado en el motor de base de datos, como PostgreSQL.



Generación automática de tablas desde Django (ORM)

Django utiliza un patrón llamado ORM (Object-Relational Mapping) que permite trabajar con bases de datos relacionales a través de modelos de datos en código Python, sin necesidad de escribir manualmente sentencias SQL.

- Cada modelo en Django es una clase Python que representa una tabla de base de datos.
- Los atributos de la clase son columnas de esa tabla, definidas con tipos de datos específicos.

- Las relaciones entre modelos (ForeignKey, ManyToManyField, OneToOneField) crean restricciones y asociaciones entre tablas.

Proceso de creación automática:

1. Definición de modelos: El desarrollador escribe clases que heredan de django.db.models.Model con sus campos.
2. Migraciones: Django compara el estado actual del modelo con el de la base de datos y genera scripts de migración que contienen instrucciones SQL para crear o modificar tablas.
3. Aplicación de migraciones: Al ejecutar python manage.py migrate, Django ejecuta las instrucciones SQL en la base de datos para crear las tablas, índices, claves foráneas y restricciones.

Ventajas del ORM en Django:

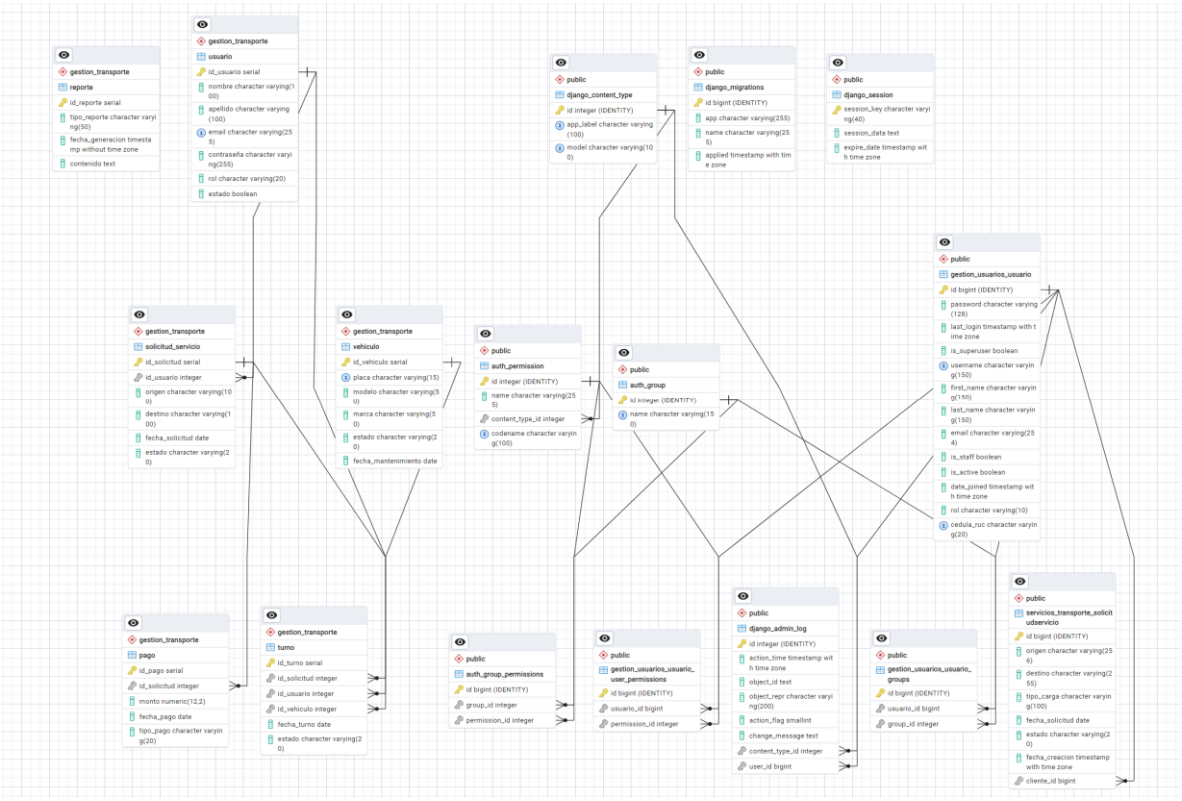
- Abstracción: Los desarrolladores trabajan con clases y objetos en Python, lo que simplifica el desarrollo.
- Portabilidad: El mismo código puede funcionar con diferentes motores de bases de datos (PostgreSQL, MySQL, SQLite) sin cambiar el código SQL.
- Mantenimiento: Facilita la evolución del modelo de datos, ya que Django genera y aplica migraciones automáticamente.
- Seguridad: Previene inyecciones SQL y facilita validaciones.

Migraciones en Django: Control de cambios en la base de datos

Las migraciones son archivos generados por Django que describen los cambios en el esquema de la base de datos, como crear tablas, agregar o modificar columnas y relaciones.

- Se generan con el comando makemigrations que crea scripts que reflejan las modificaciones del modelo.
- Se aplican con migrate, que ejecuta las sentencias SQL en la base de datos.
- Permiten versionar la base de datos, facilitando el trabajo en equipo y despliegues.

Esquema de las tablas principales para el proyecto



Arquitectura de la aplicación

¿Qué es la arquitectura de software?

La arquitectura de software es la estructura o esquema fundamental que organiza los componentes de un sistema, sus relaciones y cómo interactúan para cumplir con los requisitos funcionales y no funcionales. Es un plano que guía el diseño, desarrollo, mantenimiento y evolución del sistema. Una buena arquitectura garantiza que el sistema sea escalable, mantenible, seguro, eficiente y que permita integrar nuevas funcionalidades de forma controlada.

¿Qué es el patrón MVC?

El patrón MVC es un paradigma de arquitectura que separa la aplicación en tres componentes principales:

- **Modelo:** Gestiona la lógica de negocio y los datos (acceso a la base, reglas, operaciones).
- **Vista:** Presenta la interfaz al usuario, mostrando datos y recibiendo interacción.
- **Controlador:** Actúa como intermediario que recibe entradas del usuario, procesa la lógica usando el modelo y selecciona la vista a mostrar.

Esta separación mejora la organización, facilita el mantenimiento, promueve la reutilización y permite que diferentes equipos trabajen simultáneamente en distintas capas.

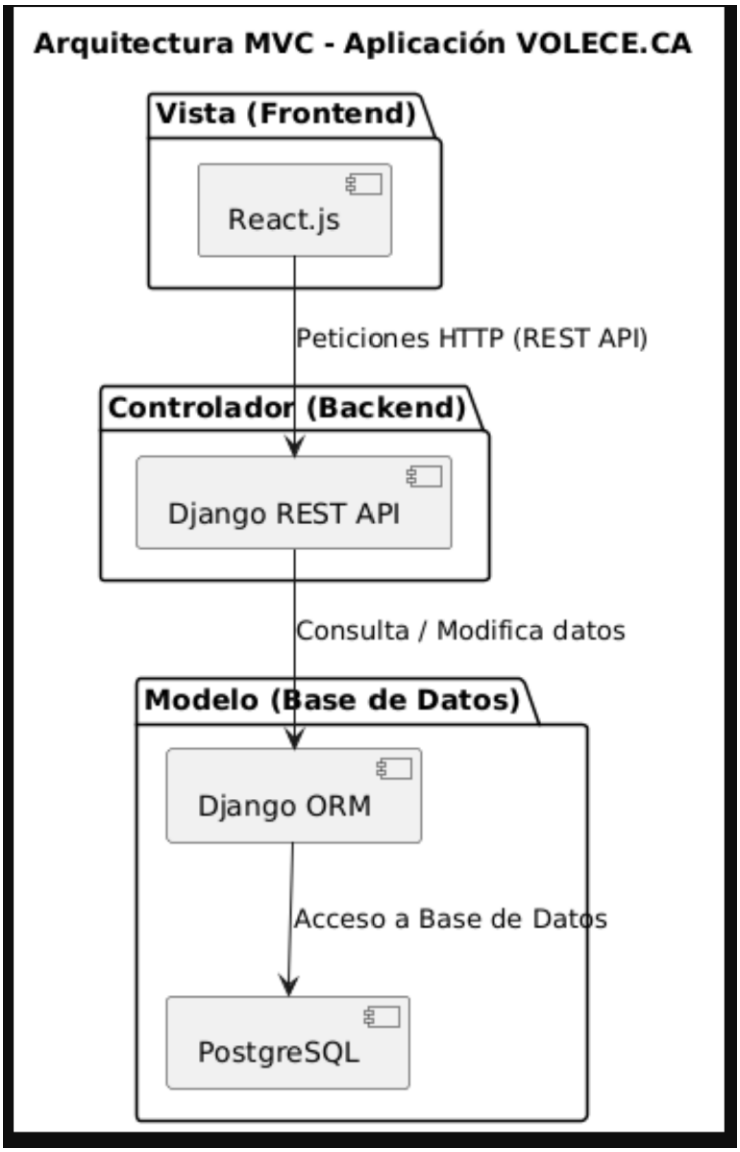
Arquitectura propuesta para VOLECE.CA TRUCKS

Componente	**Descripción y tecnologías usadas**
Modelo	Implementado con Django ORM y base de datos PostgreSQL. Define entidades como Usuario, Vehículo, SolicitudServicio, Turno, Pago y Reporte. Contiene la lógica de negocio relacionada a datos y validaciones.
Vista	Frontend desarrollado con React.js. Interfaz gráfica para usuarios (clientes, transportistas, administradores). Visualiza datos recibidos del controlador y envía eventos de usuario.
Controlador	Backend con Django REST Framework. Recibe peticiones HTTP desde la Vista (frontend). Procesa las solicitudes, aplica lógica del negocio y consulta o modifica el Modelo. Devuelve respuestas (JSON) que la Vista consume para mostrar resultados o estados.

Flujo típico en MVC para el aplicativo VOLECE.CA TRUCKS

1. El usuario interactúa con la Vista (React): Solicita un servicio, consulta turnos, etc.
2. La Vista envía la solicitud al Controlador (Django API): Mediante una llamada HTTP REST.
3. El Controlador procesa la petición: Aplica reglas de negocio, consulta o modifica datos en el Modelo.
4. El Modelo accede a la base de datos: Retorna resultados o ejecuta cambios.
5. El Controlador prepara la respuesta: Normalmente JSON con los datos o estados.
6. La Vista recibe la respuesta: Actualiza la interfaz para reflejar la información al usuario.

Gráfico del modelo MVC



Beneficios

- Separación clara de responsabilidades: Facilita el desarrollo, prueba y mantenimiento.
- Reutilización: Modelos y lógica pueden reutilizarse con diferentes interfaces si se desea.
- Escalabilidad: Permite añadir nuevas vistas o cambiar tecnologías sin afectar el modelo o controladores.
- Mejora la colaboración: Equipos frontend y backend pueden trabajar de forma independiente.

Módulos en la arquitectura MVC de la aplicación VOLECE TRUCKS

Módulo	**Ubicación MVC**	**Función principal**
Autenticación	Controlador/ Modelo	Validación de usuarios, roles y permisos.
Gestión de Usuarios	Modelo/ Controlador	Registro, actualización y consulta de usuarios.
Solicitud de Servicios	Modelo/ Controlador	Creación y seguimiento de solicitudes de transporte.
Asignación de Turno	Modelo/ Controlador	Algoritmos IA para asignar turnos equitativos.
Gestión de Vehículos	Modelo/ Controlador	Registro y estado de vehículos, mantenimientos.
Gestión Financiera	Modelo/ Controlador	Registro y control de pagos y movimientos financieros.
Visualización/ Reportes	Vista	Presentación gráfica y tabular de reportes y estados del sistema.