**Arquitectura Propuesta: Microservicios con Django y MySQL**

**Foco:** Aprovechar Django (Python) para cada microservicio, con MySQL como base de datos principal, pero con flexibilidad para otras DBs si es necesario.

**1. Estructura de Microservicios con Django**

Cada microservicio es un **proyecto Django independiente**, desplegado en contenedores Docker:

**A. Auth-Service**

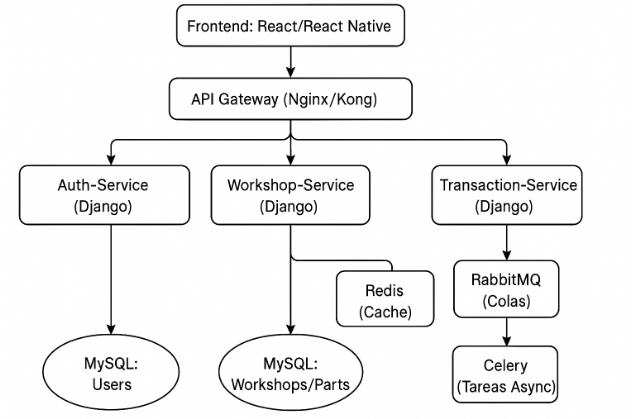
* **Función:** Registro, login, JWT y gestión de perfiles.

**B. Workshop-Service**

* **Función:** Búsqueda de talleres/repuestos, geolocalización y stock.

**C. Transaction-Service**

* **Función:** Reservas, reseñas y notificaciones.



**2. Componentes Clave**

**API Gateway Nginx**

**Base de Datos**

* **MySQL:** Un cluster por microservicio (o schemas separados si usan una instancia).

**Comunicación entre Servicios**

* **RESTful APIs** (para operaciones síncronas).

**3. Recomendaciones Clave**

**Para Django en Microservicios**

**Despliegue**

* **Docker:** Cada microservicio en su propio contenedor

**4. Alternativa si MySQL es Limitante**

Si necesitan **geolocalización avanzada**, reemplacen MySQL por **PostgreSQL + PostGIS** en el Workshop-Service

**Arquitectura Monolítica para MotusCar**

\*(Django + PostgreSQL + Docker)\*

Diagrama

Descripción generada automáticamente

1. **Django (Backend):**
   * Apps modulares: users, workshops, transactions.
   * API REST con DRF.
   * Autenticación JWT.
2. **PostgreSQL:**
   * Base de datos principal.
   * Extensión **PostGIS** para consultas geoespaciales.
3. **Redis:**
   * Caché para búsquedas frecuentes.
4. **Celery + RabbitMQ:**
   * Tareas asíncronas (ej: envío de emails).
5. **Docker:**
   * Contenedores para Django, PostgreSQL y Redis.
   * docker-compose.yml para orquestación.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Flujo de Datos:**

1. Frontend consulta APIs de Django.
2. Django interactúa con PostgreSQL (datos) y Redis (caché).
3. Tareas largas (ej: notificaciones) se delegan a Celery.