Portada

Gestionkoal – Manual de Uso
Fecha: 2025-09-12 19:45

Incluye:
- Estado del proyecto
- Arranque local (prod/dev)
- Trabajo con IA (Gemini)
- Git y despliegue Render
- Solución rápida de problemas

Índice

1. Resumen del proyecto

Estado del Proyecto — Gestionkoal (Gestión de Avisos)
Fecha: 2025-09-12 19:45
Ruta local (Windows): `C:\proyecto\gestion_avisos`
Repo GitHub: `https://github.com/infogrupokoal-ship-it/Gestionkoal.git`
Objetivo: Ejecutar en local con SQLite y desplegar en Render (Disk), manteniendo compatibilidad con Postgres.

---

1) Resumen ejecutivo
- Proyecto fuera de OneDrive (menos bloqueos y mejor rendimiento).
- Código preparado para **SQLite** (local / Render Disk) y **Postgres** (Render) con `get_db_connection()`.
- Adaptador SQL: usar **`?`** en consultas; `_execute_sql(...)` adapta a SQLite/Postgres.
- Scripts de arranque:
  - **Producción local**: `start_local_fixed.bat` → lanza `waitress` (estable, como Render).
  - **Desarrollo**: `start_dev_fixed.bat` → `flask run` con **autoreload**.
- Archivos de despliegue listos: **Procfile**, **render.yaml**, **README_RENDER.md**.

---

2) Hecho (DONE)
- [x] Arreglo de `/login` y `user_loader` para SQLite/Postgres.
- [x] Unificación de placeholders con `_execute_sql()` y `get_cursor()`.
- [x] CLI `init-db` operativo (lee `schema.sql` por bloques).
- [x] `.gitignore` recomendado (`.venv/`, `database.db`, `uploads/`).
- [x] Guías locales (README_LOCAL.md) y **comandos para IA (Gemini)**.

---

3) Pendiente inmediato (TODO)
1. Verificar **login** con usuarios de ejemplo (`password123`).
2. Subir cambios a GitHub tras confirmar que arranca:
   ```powershell
   git add -A
   git commit -m "Arranque estable; login OK; guías y scripts"
   git push
   ```
3. Desplegar en Render (SQLite + Disk) y ejecutar en Shell:
   ```bash
   FLASK_APP=app.py flask init-db
   ```

---

4) Problemas conocidos y mitigación
- **psycopg2 (Windows/Py3.13)**: no instalar en local; usar SQLite. En Render+Postgres usar `psycopg2-binary`.

1. Resumen del proyecto (cont.)

- **Herramientas Google (gaxios badRequest)**: desactivar si no se usan; si se usan, configurar credenciales/scopes correctos.
- **Cambios con waitress corriendo**: reiniciar servidor tras cambios. En desarrollo usar `start_dev_fixed.bat` (autoreload).
- **Rutas absolutas al usuario/OneDrive**: evitar; trabajar siempre en `C:\proyecto\gestion_avisos`.

---

5) Checklist de verificación
- [ ] `start_local_fixed.bat` o `start_dev_fixed.bat` levantan sin errores.
- [ ] `/login` permite entrar con usuarios de ejemplo.
- [ ] Subida/listado de archivos escribe en `UPLOAD_FOLDER`.
- [ ] `flask init-db` ejecutado en Render tras primer deploy.
- [ ] GitHub actualizado (`main` al día).

---

6) Variables de entorno útiles
- **Local (SQLite)**
  `DB_PATH=database.db`
  `UPLOAD_FOLDER=uploads`
  `FLASK_APP=app.py`
- **Render (SQLite + Disk)**
  `DB_PATH=/var/data/database.db`
  `UPLOAD_FOLDER=/var/data/uploads`
- **Render (Postgres)**
  `DATABASE_URL=postgres://...`

---

7) Flujo recomendado de trabajo
1. **Desarrollo** con `start_dev_fixed.bat` (autoreload).
2. Confirmar que funciona → **commit & push** a GitHub.
3. **Desplegar** en Render (Procfile + Disk + env vars).
4. Ejecutar `flask init-db` en Shell de Render.
5. Validar en URL pública.

---

8) Notas para IA local (Gemini)
- Directorio de trabajo: `C:\proyecto\gestion_avisos`.
- Editar solo: `app.py`, `templates/`, `static/`, `schema.sql`, `requirements.txt`.
- No editar: `.venv/`, `uploads/`, `database.db`.
- SQL: usar **`?`** como placeholder: `_execute_sql(cursor, sql, params, is_sqlite=is_sqlite)`.
- Tras cambios: si usas waitress, **reinicia**; si usas dev, autoreload aplica.

2. Arranque local

▶ Arranque diario (tras reiniciar)
Opción A – Producción local (estable, como Render):
- Doble clic: `start_local_fixed.bat`
- Equivalente manual:
  ```powershell
  cd C:\proyecto\gestion_avisos
  .venv\Scripts\activate.bat
  python run_waitress.py
  ```

Opción B – Desarrollo (autoreload, ideal si la IA edita):
- Doble clic: `start_dev_fixed.bat`
- Equivalente manual:
  ```powershell
  cd C:\proyecto\gestion_avisos
  .venv\Scripts\activate.bat
  $Env:FLASK_APP="app.py"
  $Env:FLASK_ENV="development"
  if (!(Test-Path "database.db")) { flask init-db }
  flask run
  ```

Abrir en navegador: `http://127.0.0.1:5000`

3. Trabajo con IA (Gemini)

Gemini – Comandos / Prompts Útiles

1) Mensaje de arranque (copiar/pegar al abrir la IA)
Trabaja exclusivamente en C:\proyecto\gestion_avisos.
Edita solo app.py, templates\, static\, schema.sql y requirements.txt.
No toques .venv\, uploads\ ni database.db.
Usa siempre ? como placeholder SQL y pasa is_sqlite=is_sqlite en _execute_sql.
Resume tus cambios con rutas y líneas afectadas.
Si una edición falla por texto no único, muestra el fragmento exacto que encontraste.

2) Revisión y commits
- Antes de cambios grandes: “Crea checkpoint de Git” (yo haré `git add -A && git commit -m "Checkpoint..."`).
- Al terminar: “Resume cambios y puntos de prueba”.

3) Estándares en app.py
- Conexión: `get_db_connection()` detecta DATABASE_URL o DB_PATH.
- Cursores: usa `get_cursor(conn)`.
- SQL: escribe con `?` y llama `_execute_sql(cursor, sql, params, is_sqlite=is_sqlite)`.
- Nada de `with conn.cursor()` en SQLite; usa try/finally si es necesario.

4) Plantillas
- Añadir JS no intrusivo y sin romper Jinja.
- Mantener IDs únicos y usar `DOMContentLoaded`.

5) Evitar
- Rutas absolutas a OneDrive o C:\Users\...
- Cambiar `.gitignore`, `.venv/`, `uploads/`, `database.db`.
- Introducir `%s` en SQL sin pasar por el adaptador.

4. Git y despliegue

⬆ Subir cambios a GitHub (cuándo y cómo)
Cuándo: al final del día, antes de deploy, tras arreglar algo que funciona, o antes de probar algo arriesgado.
Cómo:
```powershell
cd C:\proyecto\gestion_avisos
git status
git add -A
git commit -m "Mensaje claro del cambio"
git push
```
Si el remoto no está configurado (solo 1 vez):
```powershell
git remote add origin https://github.com/infogrupokoal-ship-it/Gestionkoal.git
git branch -M main
git push -u origin main
```

.gitignore recomendado
```
.venv/
database.db
uploads/
__pycache__/
*.pyc
```

☁ Despliegue Render (SQLite + Disk)
1. Conectar repo GitHub → New Web Service.
2. Build: `pip install -r requirements.txt`
3. Start: `python run_waitress.py`
4. Disk: montado en `/var/data`
5. Env vars:
```
DB_PATH=/var/data/database.db
UPLOAD_FOLDER=/var/data/uploads
```
6. Tras primer deploy (Shell de Render):
```
FLASK_APP=app.py flask init-db
```

4. Git y despliegue (cont.)

□ Solución rápida de problemas
- **No arranca**: venv activo, deps instaladas, `FLASK_APP=app.py`, ejecuta en la carpeta del proyecto.
- **Login falla**: usa usuarios de ejemplo (`password123`) o reemplaza `app.py` por el corregido.
- **Cambios IA no se aplican**: si estás con waitress, reinicia; con flask run, espera el autoreload.
- **psycopg2 error en Windows**: no lo uses en local. En Render/Postgres usa `psycopg2-binary`.