VoiceSurveyAgent — Sanity Check Runbook (REQ-001 … REQ-024)

Documento operativo per validazione rapida, stabile e ripetibile dei sorgenti promossi e delle integrazioni (DB, OIDC, Twilio, LLM).

1. Obiettivo e principi

Questo runbook ti guida in una sanity check completa, incrementale e ripetibile. L'idea e' partire da un runtime "dry-run" (senza side effects) e arrivare gradualmente all'end-to-end reale: DB + OIDC + Twilio + LLM, fino alla chiamata effettiva.

- Non cambiare firme dei metodi pubblici nei sorgenti promossi: se serve, aggiungere wrapper/facade, non rimuovere async.
- Zero dipendenze inutili: preferire parsing/utility in stdlib e lazy import per SDK pesanti.
- Ogni step deve lasciare un artefatto verificabile: log, output comandi, oppure una risposta HTTP attesa.
- Preferire smoke test che falliscono "puliti" (401/403/400/204) e mai 500.

2. Pre-requisiti

2.1 Struttura e ambiente

- Eseguire i comandi dalla root del repo (dove esiste la cartella src/).
- Usare sempre PYTHONPATH="src" per allinearsi al runtime reale.
- Avere una virtualenv attiva con le dipendenze installate.

2.2 Variabili d'ambiente minime (template)

Crea un file .env (o esporta variabili) per gestire modalita' e integrazioni. Esempio:

```
# Runtime
APP_ENV=local
DEBUG=0

# Database (reale o locale)
DATABASE_URL=postgresql+asyncpg://USER:PASS@HOST:5432/DBNAME

# OIDC / IdP
OIDC_ISSUER_URL=https://<issuer>/.well-known/openid-configuration
OIDC_CLIENT_ID=<client-id>
OIDC_CLIENT_SECRET=<client-secret>
OIDC_REDIRECT_URI=http://127.0.0.1:8880/api/auth/callback
OIDC_SCOPES=openid profile email

# Telephony (Twilio)
TELEPHONY_PROVIDER=twilio
```

```
TWILIO_ACCOUNT_SID=<sid>
TWILIO_AUTH_TOKEN=<token>
TWILIO_FROM_NUMBER=+1xxxxxxxxxx

# LLM
LLM_PROVIDER=openai
OPENAI_API_KEY=<key>
OPENAI_MODEL=gpt-4o-mini  # esempio

# Modalita' safety / dry-run (consigliata per smoke)
DRY_RUN=1
```

3. Gate di validazione (ripetibili)

Eseguire nell'ordine. Se un gate fallisce, fermarsi e correggere prima di proseguire.

Gate 1 — Compile

```
PYTHONPATH=src python -m compileall -q src && echo "COMPILE_OK"
```

Outcome atteso: COMPILE_OK

Gate 2 — Import-all (best effort)

```
PYTHONPATH=src python - <<'PY'
import pkgutil, importlib, traceback, sys
pkg=importlib.import_module("app")
prefix="app."
failed=[]
mods=list(pkgutil.walk_packages(pkg.__path__, prefix))
for m in mods:
    try: importlib.import_module(m.name)
    except Exception as e:
        failed.append((m.name, e))
        print("\n[FAIL]", m.name, "->", repr(e))
        traceback.print_exc(limit=4)
print("\nTOTAL:", len(mods), "FAILED:", len(failed))
sys.exit(2 if failed else 0)
PY
```

Outcome atteso: FAILED: 0

Gate 3 — Test suite

Se esiste una suite su src/ o in repo:

```
PYTHONPATH=src pytest -q
```

Outcome atteso: green. Se non ci sono test, questo gate e' informativo.

Gate 4 — Dependency sanity

```
pip check  # oppure: uv pip check
```

Outcome atteso: No broken requirements found.

Gate 5 — Dry-Run service gate (uvicorn + smoke)

```
PYTHONPATH=src uvicorn app.main:app --host 127.0.0.1 --port 8880 --log-level
info

# In un secondo terminale:
curl -sf http://127.0.0.1:8880/openapi.json >/dev/null && echo OPENAPI_OK
curl -sf http://127.0.0.1:8880/health      >/dev/null && echo HEALTH_OK
```

Outcome atteso: OPENAPI_OK e HEALTH_OK

4. Smoke live ‘no side effects’

Endpoint scelti per validare routing + auth senza attivare integrazioni esterne.

Set consigliato (3–6)

```
# 1) Health
curl -s -i http://127.0.0.1:8880/health

# 2) OpenAPI
curl -s -o /dev/null -w "%{http_code}\n" http://127.0.0.1:8880/openapi.json

# 3) Auth me (senza token -> 401)
curl -s -o /dev/null -w "%{http_code}\n" http://127.0.0.1:8880/api/auth/me

# 4) Campaigns list (senza token -> 401)
curl -s -o /dev/null -w "%{http_code}\n" http://127.0.0.1:8880/api/campaigns

# 5) Exclusions list (senza token -> 401)
curl -s -o /dev/null -w "%{http_code}\n" http://127.0.0.1:8880/api/exclusions
```

5. Progressive sanity check: da dry-run a end-to-end reale

Segui questi livelli. Passa al successivo solo se il livello corrente e’ stabile.

Livello L0 — Runtime base (no external)

1. Esegui Gate 1–5.
2. Verifica smoke live no side effects (401/200 attesi).
3. Salva evidenza: output Gate + lista OpenAPI (paths).

Livello L1 — DB reale (read-only per sanity)

Obiettivo: verificare che l’app si connetta al DB e che le query base funzionino senza mutate data.

4. Configura DATABASE_URL verso un DB di test (non produzione).
5. Avvia l’app e richiama un endpoint che fa read (se disponibile).
6. Se l’app ha migrazioni, applicarle su DB di test prima dell’avvio.

Nota: per sanity, preferire query read-only. Se un endpoint scrive, usare dataset di test e pulizia (transaction/rollback o schema dedicato).

Livello L2 — OIDC reale (login/callback)

7. Configura OIDC_* verso il tuo IdP (issuer, client id/secret, redirect).
8. Apri /api/auth/login e verifica che ritorni una authorization URL valida.
9. Completa il redirect e verifica /api/auth/callback (session/token).
10. Verifica /api/auth/me con token: deve tornare 200 con dati utente.

Outcome atteso: endpoints auth passano da 401 a 200 con token valido.

Livello L3 — Twilio reale (webhook + outbound)

11. Configura TWILIO_* e TELEPHONY_PROVIDER=twilio.
12. Verifica che il webhook /webhooks/telephony/events risponda 200/204 con payload Twilio.
13. Configura Twilio Webhook URL verso il tuo endpoint pubblico (tunnel tipo ngrok se in locale).
14. Esegui una chiamata outbound (se esiste endpoint/flow interno) verso un numero di test.

Nota safety: usare numeri Twilio verified, ambienti di test, e limiti di costo.

Livello L4 — LLM reale (dialogue)

15. Configura LLM_PROVIDER e chiave (es. OPENAI_API_KEY).
16. Verifica che la parte LLM non venga inizializzata a import-time (lazy).
17. Esegui un flusso che genera almeno 1 risposta controllata (prompt deterministico).
18. Imposta timeouts e retry limitati; loggare request id e latenza.

Livello L5 — End-to-end: campagna -> contatti -> chiamata -> dialogo -> outcome

19. Crea una campagna (API o seed).
20. Carica contatti (upload) su dataset di test.
21. Avvia lo scheduler/calls (anche manualmente) per generare tentativi di chiamata.
22. Esegui chiamata Twilio e ricevi eventi webhook.
23. Esegui turn LLM e persisti outcome nel DB.
24. Verifica consistenza: campaign status, attempts, exclusions aggiornate.

6. Checklist per ogni nuova evoluzione (REQ-014..REQ-024 e oltre)

Per ogni REQ nuovo/promosso, applicare sempre questa mini-checklist:

25. Aggiorna OpenAPI e confronta: nuove route presenti? prefissi corretti?
26. Esegui Gate 1–5.
27. Aggiungi 1 smoke endpoint per la feature nuova (anche se risponde 401/400).

28. Se la feature dipende da provider esterni, introdurre modalita' DRY_RUN per evitare side effects.
29. Aggiungi almeno 1 contract test in-memory per interfacce provider/handler (evita mismatch di firme).

7. Troubleshooting rapido

7.1 404 Not Found su un endpoint atteso

- Controlla che il router sia incluso in app.main (include_router).
- Controlla prefix e path nel router; confronta con OpenAPI.

7.2 500 su webhook/provider

- Leggi stacktrace: spesso e' dependency (provider) o mismatch di firma.
- Aggiungi fallback provider/mock per DRY_RUN.
- Gestisci parsing payload senza dipendenze opzionali non installate.

7.3 401/403 inattesi

- Verifica middleware auth e configurazione OIDC.
- Verifica header Authorization e token.