# Digantara - Ground Pass Prediction

## Explanatory Notes & Architecture

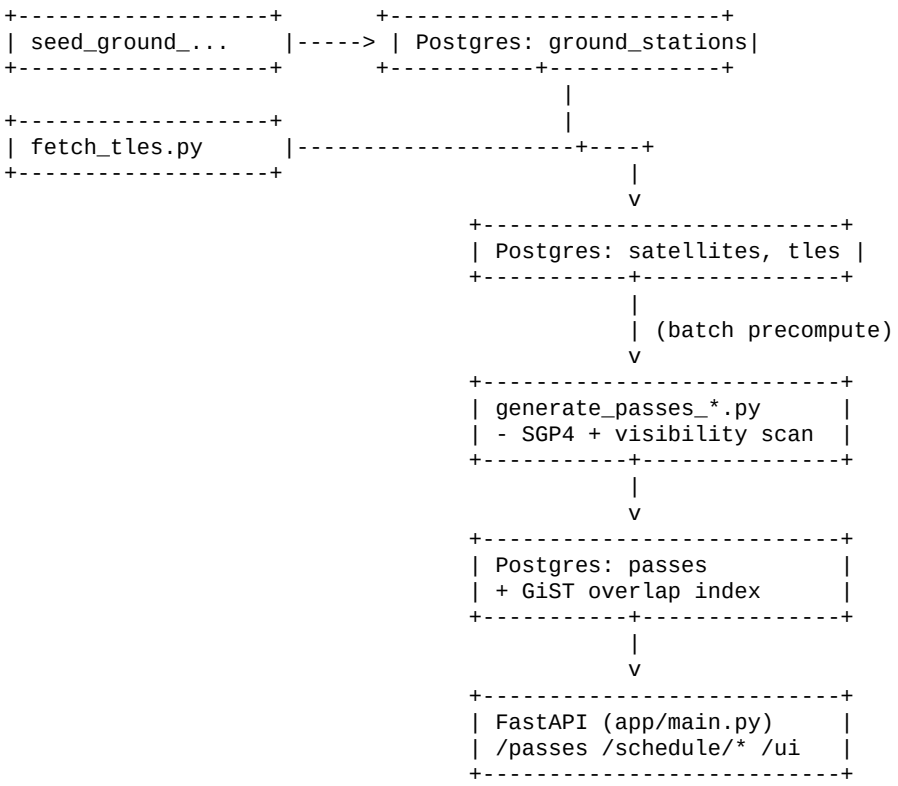
## What this project does

This backend predicts **satellite passes** over a ground station (when the satellite is visible). It works in 4 simple steps: (1) download latest TLEs, (2) use SGP4 to simulate the orbit for the next 7 days, (3) detect when the satellite is above the station's horizon (start/end time), and (4) save the pass windows to PostgreSQL so APIs can answer quickly.

### Why we store passes in the DB

If we compute passes on every API call, requests will be slow. So we precompute passes once (batch) and store them. Then the API mostly does fast database reads.

## High-level architecture diagram



## End-to-end flow

• **A. Seed stations**: load 50 ground stations from CSV into Postgres.

• **B. Fetch TLEs**: download active satellite TLEs from CelesTrak and store them.

• **C. Generate passes**: for each (satellite, station), scan time and find when elevation > 0° → store start/end.

• **D. Query passes**: API returns passes that overlap the requested time window (fast due to index).

• **E. Build schedule**: choose the best non-overlapping passes for one station (DP).

## Run commands

```
1) Start Postgres
   docker compose up -d

2) Install dependencies (Python 3.12)
   py -3.12 -m pip install -r requirements.txt

3) Create tables
   py -3.12 -m alembic upgrade head

4) Seed ground stations (50)
   py -3.12 -m app.scripts.seed_ground_stations

5) Fetch TLEs (example: 200 active objects)
   py -3.12 -m app.scripts.fetch_tles --group active --limit 200

6) Generate passes (demo run)
   py -3.12 -m app.scripts.generate_passes_7d_batched `
   --days 2 --gs-limit 50 --sat-limit 30 `
   --chunk-hours 24 --step 60 --delete-existing

7) Run API
   py -3.12 -m uvicorn app.main:app --reload

8) Test (Swagger)
   http://127.0.0.1:8000/docs
```

# Main folders/files (simple map)

**app/main.py** - FastAPI server: pass queries + scheduling endpoints + small demo UI.

**app/db/conn.py** - DB connection helper (connect, commit/rollback, close).

**app/scripts/seed_ground_stations.py** - Load the 50 stations CSV into DB.

**app/scripts/fetch_tles.py** - Fetch latest TLEs from CelesTrak and store to DB.

**app/scripts/generate_passes_7d_batched.py** - Batch job to compute passes for next days and insert to DB.

**app/orbit/pass_prediction.py** - Pass detection logic (scan + refine start/end).

**app/orbit/visibility.py** - Math to compute elevation angle at a station.

**app/schedule/optimizer.py** - Best non-overlapping schedule using dynamic programming.

**alembic/versions/*** - Database schema migrations + indexes.

# Database tables

| Table | Stores | Key columns |
|---|---|---|
| ground_stations | Station list | id, code, name, lat, lon, alt_m |
| satellites | Satellite identity | id, norad_id, name |
| tles | Latest + historical TLEs | satellite_id, line1, line2, epoch, fetched_at |
| passes | Predicted pass windows | satellite_id, ground_station_id, start_ts, end_ts, duration_s, max_elev_deg |

## Why /passes query is fast

We store each pass as a time interval [start_ts, end_ts]. When the user asks for a time window, we return passes that overlap that window. A GiST index on the time range makes this fast even for large tables.

```
WHERE ground_station_id = :gs_id
  AND tstzrange(start_ts, end_ts, '[)') && tstzrange(:start, :end, '[)')
```

# API endpoints

• **GET /health** - Server health check.

• **GET /db/health** - DB health check.

• **GET /ground-stations?limit=50** - List ground stations.

• **GET /passes?gs_id=1&start;=...&end;=...** - List passes overlapping the time window.

• **GET /schedule/best?gs_id=...&start;=...&end;=...&metric;=...** - Best non-overlapping schedule for one station.

• **GET /schedule/top?gs_id=...&start;=...&end;=...&k;=5** - Top K passes by metric (no non-overlap).

• **GET /network/schedule/best?start=...&end;=...** - Best schedule per station + network summary.

• **GET /ui** - Small UI to try endpoints quickly.

### Scheduling (best non-overlapping passes)

For one station, passes can overlap in time. The station can track only one satellite at a time, so we choose a set of passes that do not overlap. We maximize a score like total duration or max elevation. This is solved with a standard DP approach called Weighted Interval Scheduling (about O(n log n)).

# Performance notes

• Passes are precomputed in batch scripts, so the API mostly reads from DB.

• The overlap index helps keep /passes fast for large datasets.

• Generation uses chunking (e.g., 24h chunks) to keep memory stable.

• Scheduling runs on the query result set; you can cap rows if needed.

### Assumptions

• Visibility uses an approximate TEME→ECEF conversion (good enough for assignment-level pass windows).

• Pass detection uses coarse scan + refinement (trade speed vs accuracy).

• Rate limiting is in-memory; production would use Redis.