GIS Script Generator
User Guide

PyQGIS | ArcPy | QGIS Project | ArcGIS Toolbox
Folium | Kepler.gl | pydeck | Web UI
Catalogue-driven script generation from PostGIS

Version 0.1.0 | 2026-02-28

This guide covers all eight template types generated by gis-codegen and gis-catalogue, the Web UI, the full operations reference, the symbology dispatch table, and complete CLI documentation.

1. Project Overview

gis-codegen is a command-line tool that connects to a PostGIS database, extracts spatial layer metadata (geometry type, SRID, columns, primary keys, row counts), and generates ready-to-run scripts or project files for eight GIS platforms.

gis-catalogue reads a map catalogue Excel file and writes one script per map entry, auto-selecting the correct renderer based on the symbology_type column.

gis-ui launches a browser-based form that connects, extracts, and returns the generated file as a download -- no command line required.

Template types

CLI command	Template type	Output file
gis-codegen --platform pyqgis	PyQGIS standalone script	*.py
gis-codegen --platform arcpy	ArcPy / ArcGIS Pro script	*.py
gis-codegen --platform folium	Folium / Leaflet HTML map	*.py
gis-codegen --platform kepler	Kepler.gl HTML map	*.py
gis-codegen --platform deck	pydeck / deck.gl HTML map	*.py
gis-codegen --platform export	PostGIS -> GeoPackage script	*.py
gis-codegen --platform qgs	QGIS project file	*.qgs
gis-codegen --platform pyt	ArcGIS Python Toolbox	*.pyt
gis-catalogue --platform pyqgis	Catalogue PyQGIS (x N maps)	M##_name.py
gis-catalogue --platform arcpy	Catalogue ArcPy (x N maps)	M##_name.py

Package layout

```
src/gis_codegen/
    __init__.py         Public API: connect, extract_schema, generate_*
    extractor.py        PostGIS metadata queries
    generator.py        Code generation (all 8 platforms)
    catalogue.py        Excel-driven per-map code generation
    cli.py              gis-codegen CLI entry point
    app.py              Flask web UI (gis-ui entry point)
tests/
    conftest.py         Shared fixtures
    test_generator.py   test_catalogue.py   test_extractor.py
    test_app.py         test_integration.py
.github/workflows/
    ci.yml              Unit + integration CI jobs
maps/                   Generated output (git-ignored)
make_pdf.py             This PDF generator
```

2. Installation & Setup

Install

```
# From the project root:
pip install -e .

# With web mapping extras (folium, kepler, pydeck):
pip install -e ".[web]"

# With Flask web UI:
pip install -e ".[server]"

# With dev tools (pytest, coverage, openpyxl):
pip install -e ".[dev]"

# With PostGIS integration tests (requires Docker):
pip install -e ".[integration]"

# Entry points after install:
gis-codegen   --help
gis-catalogue --help
gis-ui                  # launches web UI at http://localhost:5000
```

Required environment variable

PGPASSWORD must always be set before running any command. It is never stored in config files or embedded in generated scripts.

```
# Windows
set PGPASSWORD=your_password

# Linux / macOS
export PGPASSWORD=your_password
```

Optional environment variables

Variable	Default	Description
PGHOST	localhost	Database host
PGPORT	5432	Database port
PGDATABASE	my_gis_db	Database name
PGUSER	postgres	Database user
PGPASSWORD	(required)	Database password -- no default

Config file (gis_codegen.toml)

Place in the project root, or pass with --config. CLI flags override it.

```
[database]
host   = "localhost"
port   = 5432
dbname = "my_gis_db"
user   = "postgres"
# password NOT stored here -- use PGPASSWORD env var
```

```
[defaults]
platform      = "pyqgis"
schema_filter = "public"
no_row_counts = false
output        = "output.py"
save_schema   = "schema.json"
```

3. Schema Extraction

The extractor queries geometry_columns, information_schema.columns, information_schema.table_constraints, and pg_class to build a full metadata snapshot of all spatial layers in the database.

Preview layers (no files written)

```
gis-codegen --list-layers

# Example output:
#  schema  table    geom_type        srid   rows
#  ------  -----    ---------        ----   ----
#  public  parcels  MULTIPOLYGON     4326   ~1 000
#  public  roads    MULTILINESTRING  4326   ~500
```

Save schema JSON for offline use

```
gis-codegen --save-schema schema.json
# Then generate without a live DB connection:
gis-codegen --platform pyqgis -i schema.json -o my_map.py
```

Schema JSON structure

```
{
  "database": "my_gis_db",
  "host": "localhost",
  "layer_count": 1,
  "layers": [
    {
      "schema": "public",
      "table": "parcels",
      "qualified_name": "public.parcels",
      "geometry": {"column":"geom","type":"MULTIPOLYGON","srid":4326},
      "columns": [
        {"name":"parcel_id","data_type":"integer","nullable":false},
        {"name":"address",  "data_type":"character varying","nullable":true}
      ],
      "primary_keys": ["parcel_id"],
      "row_count_estimate": 1000
    }
  ]
}
```

4. PyQGIS Template

PyQGIS scripts connect to PostGIS via QgsDataSourceUri and load layers into a QGIS project. They can run as standalone scripts (outside QGIS) or be pasted into the QGIS Python console.

Generate

```
# From a live database:
gis-codegen --platform pyqgis -o my_map.py

# From a saved schema:
gis-codegen --platform pyqgis -i schema.json -o my_map.py

# Filter to one layer:
gis-codegen --platform pyqgis --layer public.parcels -o parcels.py

# Add operation blocks:
gis-codegen --platform pyqgis --op buffer --op dissolve -o parcels.py
```

Template anatomy

```
"""
Auto-generated PyQGIS script
Database : my_gis_db @ localhost:5432
Generated: 2026-02-23 14:00   Layers: 2
"""

import os, sys
from qgis.core import (
    QgsApplication, QgsDataSourceUri, QgsVectorLayer, QgsProject,
    QgsCoordinateReferenceSystem,
)

# Remove the next 2 lines if running inside the QGIS console:
qgs = QgsApplication([], False)
qgs.initQgis()

DB_HOST = "localhost"
DB_PORT = "5432"
DB_NAME = "my_gis_db"
DB_USER = "postgres"
DB_PASSWORD = "..."         # value at generation time

# ================================================================
# Layer : public.parcels   Geom: MULTIPOLYGON   SRID: 4326
# ================================================================
uri_parcels = QgsDataSourceUri()
uri_parcels.setConnection(DB_HOST, DB_PORT, DB_NAME, DB_USER, DB_PASSWORD)
uri_parcels.setDataSource("public", "parcels", "geom", "", "parcel_id")

lyr_parcels = QgsVectorLayer(uri_parcels.uri(False), "parcels", "postgres")
if not lyr_parcels.isValid():
    print("[ERROR] Layer 'parcels' failed to load.")
else:
    QgsProject.instance().addMapLayer(lyr_parcels)
    print(f"[OK] parcels: {lyr_parcels.featureCount()} features")

qgs.exitQgis()   # remove if running inside QGIS console
```

How to run

- Standalone: set PGPASSWORD=... then python my_map.py
- (requires the QGIS Python environment on PATH)
- QGIS console: paste the script body; remove QgsApplication init/exit lines
- QGIS Processing Toolbox: Plugins -> Python Console -> Open Script

5. ArcPy Template

ArcPy scripts create a temporary .sde connection file, access PostGIS feature classes through that connection, and optionally run analysis tools. They require ArcGIS Pro with the PostgreSQL client libraries installed.

Generate

```
gis-codegen --platform arcpy -o my_map.py

# With 3D massing ops:
gis-codegen --platform arcpy --op extrude --op scene_layer -o massing.py
```

Template anatomy

```
import arcpy, os, tempfile

DB_HOST = "localhost"
DB_PORT = "5432"
DB_NAME = "my_gis_db"
DB_USER = "postgres"
DB_PASSWORD = "..."

SDE_FOLDER = tempfile.gettempdir()
SDE_FILE   = os.path.join(SDE_FOLDER, f"{DB_NAME}.sde")
if not os.path.exists(SDE_FILE):
    arcpy.management.CreateDatabaseConnection(
        database_platform="POSTGRESQL",
        instance=f"{DB_HOST},{DB_PORT}",
        account_authentication="DATABASE_AUTH",
        username=DB_USER, password=DB_PASSWORD,
        save_user_pass="SAVE_USERNAME", database=DB_NAME, ...
    )

fc_parcels = os.path.join(SDE_FILE, "public.parcels")
if arcpy.Exists(fc_parcels):
    count = int(arcpy.management.GetCount(fc_parcels)[0])
    print(f"[OK] parcels: {count} rows")
```

How to run

- ArcGIS Pro Python console: paste and run directly
- Standalone: run from the ArcGIS Pro Python env (arcgispro-py3)
- Script Tool: add as a Python Script Tool in a Toolbox (.atbx)

NOTE: The SDE file persists between runs. Delete it manually if you change the host, database name, or credentials.

6. QGIS Project File (--platform qgs)

Instead of a Python script, this platform emits a .qgs XML file that opens directly in QGIS 3.x. All PostGIS layers appear pre-connected in the Layers panel -- no scripting required. QGIS prompts for the database password on open; it is never embedded in the file.

Generate

```
gis-codegen --platform qgs -o project.qgs

# Then open in QGIS:
#   File -> Open Project -> select project.qgs
#   QGIS prompts for the PostGIS password
#   All layers appear in the Layers panel
```

Key design decisions

- Layer IDs are deterministic: {table}_{md5(qualified_name)[:8]} -- safe to version-control, same ID on every regeneration
- Geometry type mapping: POINT/MULTIPOINT -> Point (code 0), LINESTRING/MULTILINESTRING -> Line (code 1), POLYGON/MULTIPOLYGON -> Polygon (code 2)
- Project CRS and map canvas default to EPSG:4326 (world bounds)
- Per-layer <srs> uses minimal <authid>EPSG:{srid}</authid> -- QGIS resolves the full WKT from its internal EPSG registry
- sslmode=disable in the datasource string (change to require for production)

File structure

```
<!DOCTYPE qgis PUBLIC 'http://mrcc.com/qgis.dtd' 'SYSTEM'>
<qgis projectname="my_gis_db" version="3.28.0-Firenze">
  <projectCrs>
    <spatialrefsys><authid>EPSG:4326</authid></spatialrefsys>
  </projectCrs>
  <mapcanvas name="theMapCanvas">
    <units>degrees</units>
    <extent>-180 -90 180 90</extent>
    <destinationsrs><spatialrefsys><authid>EPSG:4326</authid></spatialrefsys></destinationsrs>
  </mapcanvas>
  <projectlayers>
    <maplayer type="vector" geometry="Polygon">
      <id>parcels_a1b2c3d4</id>
      <datasource>dbname='my_gis_db' host=localhost port=5432
        sslmode=disable key='parcel_id' srid=4326 type=Polygon
        table="public"."parcels" (geom) sql=</datasource>
      <layername>parcels</layername>
      <provider encoding="UTF-8">postgres</provider>
      <srs><spatialrefsys><authid>EPSG:4326</authid></spatialrefsys></srs>
      <layerGeometryType>2</layerGeometryType>
    </maplayer>
  </projectlayers>
  <legend>...</legend>
</qgis>
```

NOTE: --op flags are silently ignored for the qgs platform. Operations are only supported on pyqgis and arcpy.

7. ArcGIS Python Toolbox (--platform pyt)

This platform generates a .pyt Python Toolbox file. When opened in ArcGIS Pro via Insert -> Toolbox -> Add Python Toolbox, it presents a GUI dialog with connection parameters pre-filled. Running the tool loads all PostGIS layers into the active map. The password is never hardcoded -- the dialog prompts for it at runtime.

Generate

```
gis-codegen --platform pyt -o loader.pyt

# Then in ArcGIS Pro:
#   Insert -> Toolbox -> Add Python Toolbox
#   Navigate to loader.pyt
#   Double-click 'Load PostGIS Layers'
#   Fill in password, click Run
```

Toolbox structure

```
class Toolbox:
    def __init__(self):
        self.label = "PostGIS Loader"
        self.tools = [LoadPostGISLayers]

class LoadPostGISLayers:
    def getParameterInfo(self):   # 6 parameters:
        # host     (GPString, Required, default from db_config)
        # port     (GPString, Required, default from db_config)
        # dbname   (GPString, Required, default from db_config)
        # user     (GPString, Required, default from db_config)
        # password (GPStringHidden, Required, NO default)
        # schema_filter (GPString, Optional)

    def isLicensed(self):  return True
    def updateParameters(self, parameters):  pass
    def updateMessages(self, parameters):    pass

    def execute(self, parameters, messages):
        # CreateDatabaseConnection -> postgis_conn.sde in scratchFolder
        # ArcGISProject('CURRENT').activeMap.addDataFromPath per layer
        _tables = [
            ('public', 'parcels'),
            ('public', 'roads'),
            ...
        ]
```

NOTE: --op flags are silently ignored for the pyt platform. Operations are only supported on pyqgis and arcpy.

8. Web UI (gis-ui)

The web UI is a minimal Flask application that exposes the full gis-codegen pipeline as a browser form. Fill in the connection details, choose a platform, and click Generate -- the script or project file is returned as a file download. No command line or Python knowledge needed.

Start the server

```
pip install -e ".[server]"
gis-ui
# -> Serving on http://0.0.0.0:5000
# Open http://localhost:5000 in a browser
```

Routes

Method	Path	Description
GET	/	Render the connection + platform form
POST	/generate	Connect, extract, generate, return file download

File download extensions

Platform	Downloaded as
qgs	*.qgs (QGIS project file)
pyt	*.pyt (ArcGIS Python Toolbox)
pyqgis, arcpy, folium	*.py (Python script)

Security notes

- Password is submitted via POST body only -- never in the URL or query string
- Password is never stored, logged, or reflected in the response
- Jinja2 autoescaping is active (Flask default) -- XSS safe
- Connection errors re-render the form with an error message (HTTP 400); the error text is from the psycopg2 exception and does not include the password
- No authentication layer is included -- run behind a reverse proxy or firewall if exposing beyond localhost

NOTE: The web UI supports all eight platforms. --op flags are not available in the web form; use the CLI directly for operation blocks.

9. Web Mapping Templates (folium / kepler / deck)

Web templates read PostGIS via geopandas + SQLAlchemy and produce a standalone HTML file. Install extras first: pip install -e ".[web]"

NOTE: All web templates use DB_PASSWORD = os.environ["PGPASSWORD"] -- the password is NEVER embedded in generated scripts.

Folium / Leaflet (--platform folium)

```
gis-codegen --platform folium -o map.py
python map.py        # -> map.html
```

Produces a Leaflet map with GeoJson layers and tooltips from the first 5 columns per layer. Polygons, lines, and points each get a distinct colour from a 6-colour cycling palette. A LayerControl widget is added automatically. All geometries are reprojected to EPSG:4326.

Kepler.gl (--platform kepler)

```
gis-codegen --platform kepler -o kepler_map.py
python kepler_map.py     # -> kepler_map.html
                         # (or render inline in Jupyter)
```

Loads all layers into a KeplerGl map object. If a height column is detected (height, floors, elevation, z, roof_height, ...) a 3D tip comment is added. Enable it in the Kepler UI: Layers -> type -> 3D buildings, height field = detected column.

pydeck / deck.gl (--platform deck)

```
gis-codegen --platform deck -o deck_map.py
python deck_map.py   # -> deck_map.html
```

Polygon/line layers use GeoJsonLayer; point layers use ScatterplotLayer. When a height column is detected, commented extrusion lines are added -- uncomment extruded=True and set pitch=45 for a 3D view.

GeoPackage export (--platform export)

```
gis-codegen --platform export -o export.py
python export.py     # -> my_gis_db_export.gpkg
```

Dumps every spatial layer from PostGIS to a single GeoPackage file using geopandas. Each layer is written in a try/except block so one failure does not abort the rest. The script exits with code 1 if any layer failed.

Colour palette (shared by Folium and pydeck)

Layer index	Hex colour	RGBA
0 (first)	#ff8c00	[255, 140, 0, 160] orange
1	#0080ff	[0, 128, 255, 160] blue
2	#00c864	[0, 200, 100, 160] green
3	#ff3232	[255, 50, 50, 160] red
4	#b400ff	[180, 0, 255, 160] purple

5+	#00c8c8	[0, 200, 200, 160] teal (cycles)

10. Operations Reference (--op flag)

Operations inject additional code blocks into PyQGIS or ArcPy scripts. Repeat --op for multiple operations in one script:

```
gis-codegen --platform pyqgis --op buffer --op dissolve -o out.py
gis-codegen --platform arcpy  --op extrude --op scene_layer -o 3d.py
```

NOTE: Operations are not supported on web platforms (folium, kepler, deck, export) or on qgs and pyt. The --op flag is silently ignored for those platforms.

General operations (10)

--op value	PyQGIS	ArcPy
reproject	processing: native:reprojectlayer	management.Project
export	QgsVectorFileWriter -> GeoJSON	conversion.FeatureClassToShapefile
buffer	processing: native:buffer	analysis.Buffer
clip	processing: native:clip (commented)	analysis.Clip (commented)
select	selectByExpression	SelectLayerByAttribute
dissolve	processing: native:dissolve	management.Dissolve
centroid	processing: native:centroids	management.FeatureToPoint
field_calc	processing: native:fieldcalculator	management.CalculateField
spatial_join	joinattributesbylocation (commented)	analysis.SpatialJoin (commented)
intersect	native:intersection (commented)	analysis.Intersect (commented)

Operations marked (commented) produce a ready-to-uncomment template that requires you to define an input boundary or overlay layer first.

3D massing operations (5)

--op value	Description	Notes
extrude	Data-driven height extrusion renderer	PyQGIS: QgsPolygon3DSymbol ArcPy: ddd.ExtrudePolyg
z_stats	Min / max / mean Z vertex statistics	PyQGIS: hasZ + vertex loop ArcPy: ddd.AddZInformation
floor_ceiling	Extrude from base_height to roof_height field	Two-field floor-to-roof extrusion
volume	Approx. volume = footprint area x height	Fast estimate; exact: ST_Volume() in PostGIS
scene_layer	Export 3D layer package	PyQGIS: native:convert3dtiles ArcPy: CreateSceneLayerP

11. Catalogue-Driven Generation

gis-catalogue reads a map catalogue Excel file and writes one script per map entry. Only rows where status is 'have' or 'partial' AND spatial_layer_type contains 'Vector' are included.

Inclusion rules

status	spatial_layer_type	Included?
have	Vector	YES
partial	Vector	YES
have	Raster/Vector	YES (+ raster TODO note)
todo	Vector	NO -- skipped
have	Raster	NO -- skipped
todo	Raster	NO -- skipped

Key catalogue columns

Column	Role in generated script
map_id	Section header comment and layout name (e.g. M07_layout)
short_name	Python variable names and PostGIS table name (public.short_name)
spatial_layer_type	Triggers raster TODO note when 'Raster' is in the value
symbology_type	Drives automatic renderer selection (see Chapter 12)
validation_checks	Written as # [ ] item checklist
deliverable_format	Written into the export stub comment
classification	Passed to categorized renderer block as scheme comment

CLI usage

```
# Generate PyQGIS scripts (default):
gis-catalogue --input catalogue.xlsx --output-dir ./maps/

# Generate ArcPy scripts:
gis-catalogue --input catalogue.xlsx --platform arcpy --output-dir ./maps_arcpy/

# Preview what would be generated (no files written):
gis-catalogue --input catalogue.xlsx --list

# Use a saved schema JSON instead of a live DB connection:
gis-catalogue --input catalogue.xlsx --schema schema.json

# Add operation blocks to every script:
gis-catalogue --input catalogue.xlsx --op buffer --op reproject

# Override DB connection:
gis-catalogue --input catalogue.xlsx --host myserver --dbname prod_db
```

Output naming

```
# One file per included map:
```

```
maps/
    M03_occupation_sol_2026.py
    M07_hauteurs_etages_degrade.py
    M27_commerces_typologie_concentrations.py
    ...
```

12. Symbology Dispatch Table

The symbology_type cell in the catalogue drives automatic renderer selection. Matching is case-insensitive on the full cell value. The first matching rule wins.

Keyword(s) in symbology_type	PyQGIS renderer	ArcPy renderer
heatmap OR densité	QgsHeatmapRenderer	HeatMapRenderer (Pro 3.x)
réseau OR network	QgsCategorizedSymbolRenderer (QgsLineSymbol)	UniqueValueRenderer (NET_FIELD placeholde
catégoriel / catégorie (without choroplèthe)	QgsCategorizedSymbolRenderer	UniqueValueRenderer
choroplèthe + catégoriel	QgsCategorizedSymbolRenderer	UniqueValueRenderer
choroplèthe / dégradé / gradué	QgsGraduatedSymbolRenderer	GraduatedColorsRenderer
points OR polygones	QgsSingleSymbolRenderer	SimpleRenderer
(anything else)	# TODO: configure renderer	# TODO: configure renderer

NOTE: All renderer blocks use a TODO comment to mark the field name (GRAD_FIELD, CAT_FIELD, NET_FIELD) that you must verify against the actual PostGIS table schema.

13. Catalogue PyQGIS Template Anatomy

Each file generated by gis-catalogue --platform pyqgis has six sections:

1 Docstring header (16 fields)

```
"""
Map ID    : M07
Title     : Hauteurs / nombre d'etages (degrade)
Theme     : Forme urbaine > Gabarits
Objective : Representer gabarits et gradient de hauteur
Symbology : choroplethe (degrade)
Sources   : OSM building:levels  [2024-2026]
Status    : have  |  Owner: Liam
Generated : 2026-02-23 14:48
"""
```

2 QGIS imports & init

```
import os
from qgis.core import (
    QgsApplication, QgsDataSourceUri, QgsVectorLayer, QgsProject,
)
qgs = QgsApplication([], False)
qgs.initQgis()
```

3 DB credentials

```
DB_HOST = "localhost"
DB_PORT = "5432"
DB_NAME = "my_gis_db"
DB_USER = "postgres"
DB_PASSWORD = os.environ["PGPASSWORD"]   # never hardcoded
```

4 Layer load block

```
uri_hauteurs_etages_degrade = QgsDataSourceUri()
uri_hauteurs_etages_degrade.setConnection(
    DB_HOST, DB_PORT, DB_NAME, DB_USER, DB_PASSWORD)
uri_hauteurs_etages_degrade.setDataSource(
    "public", "hauteurs_etages_degrade", "geom", "", "")

lyr_hauteurs_etages_degrade = QgsVectorLayer(
    uri_hauteurs_etages_degrade.uri(False), "hauteurs_etages_degrade", "postgres")

if not lyr_hauteurs_etages_degrade.isValid():
    print("[ERROR] 'hauteurs_etages_degrade' failed to load.")
else:
    QgsProject.instance().addMapLayer(lyr_hauteurs_etages_degrade)
```

5 Auto-dispatched symbology block

```
    # --- Symbology: choroplethe (degrade) ---
    from qgis.core import (
        QgsGraduatedSymbolRenderer, QgsClassificationQuantile,
        QgsColorBrewerColorRamp,
    )
```

```
    GRAD_FIELD_... = "value"   # TODO: verify field name
    _rend = QgsGraduatedSymbolRenderer(GRAD_FIELD_...)
    _rend.setClassificationMethod(QgsClassificationQuantile())
    _rend.updateClasses(lyr_..., 5)
    _rend.updateColorRamp(QgsColorBrewerColorRamp("YlOrRd", 5))
    lyr_....setRenderer(_rend)
    lyr_....triggerRepaint()
```

6 Validation checklist & export stub

```
    # --- Validation checks ---
    # [ ] valeurs nulles
    # [ ] plausibilite niveaux
    # [ ] palette lisible

    # --- Export: Layout PDF + couche ---
    # TODO: configure a QGIS print layout named "M07_layout" then:
    # from qgis.core import QgsLayoutExporter
    # ...

qgs.exitQgis()
```

14. Catalogue ArcPy Template Anatomy

Each file generated by gis-catalogue --platform arcpy has the same six sections as the PyQGIS version but uses ArcPy idioms throughout.

1-3 Docstring, imports & credentials

```
# Docstring header identical to PyQGIS version.

import arcpy
import os
import tempfile

DB_HOST = "localhost"
DB_PORT = "5432"
DB_NAME = "my_gis_db"
DB_USER = "postgres"
DB_PASSWORD = os.environ["PGPASSWORD"]
```

4 SDE connection file

```
SDE_FOLDER = tempfile.gettempdir()
SDE_FILE   = os.path.join(SDE_FOLDER, f"{DB_NAME}.sde")

if not os.path.exists(SDE_FILE):
    arcpy.management.CreateDatabaseConnection(
        database_platform="POSTGRESQL",
        instance=f"{DB_HOST},{DB_PORT}",
        account_authentication="DATABASE_AUTH",
        username=DB_USER, password=DB_PASSWORD,
        save_user_pass="SAVE_USERNAME", database=DB_NAME, ...
    )
```

5 Feature class check and project setup

```
fc_... = os.path.join(SDE_FILE, "public.hauteurs_etages_degrade")

if arcpy.Exists(fc_...):
    count = int(arcpy.management.GetCount(fc_...)[0])
    print(f"[OK] {count} rows")

    aprx   = arcpy.mp.ArcGISProject("CURRENT")
    mp_map = aprx.listMaps()[0]
    lyr_... = mp_map.addDataFromPath(fc_...)
```

6 Auto-dispatched ArcPy symbology block

```
    # --- Symbology: choroplethe (degrade) ---
    GRAD_FIELD_... = "value"   # TODO: verify field name
    sym = lyr_....symbology
    sym.updateRenderer("GraduatedColorsRenderer")
    sym.renderer.classificationField = GRAD_FIELD_...
    sym.renderer.breakCount = 5
    lyr_....symbology = sym
    aprx.save()
```

NOTE: Use APRX_PATH = "CURRENT" when running inside the ArcGIS Pro console. For standalone scripts, set

APRX_PATH to the path of your .aprx file.

15. CLI Quick Reference

gis-codegen -- all flags

```
gis-codegen [connection] [generation] [schema]

Connection flags (override config file and env vars):
  --host HOST         Database host
  --port PORT         Database port
  --dbname DBNAME     Database name
  --user USER         Database user
  --config FILE       TOML config file path

Generation flags:
  --platform          pyqgis | arcpy | folium | kepler | deck |
                      export | qgs | pyt
  --op OPERATION      Add an operation block (repeatable)
                      See Chapter 10 for all 15 valid values
                      (ignored for folium/kepler/deck/export/qgs/pyt)
  --layer SCHEMA.TABLE  Restrict to this layer (repeatable)
  -o / --output FILE  Write to file (default: stdout)

Schema flags:
  --list-layers       Print layer table and exit
  --save-schema FILE  Save schema JSON and exit
  --no-row-counts     Skip row count queries (faster)
  --schema-filter S   Only include layers in schema S

Priority: CLI flags > config file > env vars > built-in defaults
```

gis-catalogue -- all flags

```
gis-catalogue [options]

Required:
  -i / --input FILE        Path to catalogue .xlsx file

Optional:
  -o / --output-dir DIR    Output directory  (default: ./maps/)
  -p / --platform          pyqgis | arcpy    (default: pyqgis)
  --host HOST              (default: localhost / PGHOST)
  --port PORT              (default: 5432 / PGPORT)
  --dbname DBNAME          (default: my_gis_db / PGDATABASE)
  --user USER              (default: postgres / PGUSER)
  --schema FILE            Schema JSON from gis-codegen --save-schema
                           (offline mode -- no DB connection needed)
  --op OPERATION           Add operation block to every script (repeatable)
  --list                   Print filtered maps and exit (no files written)

Filter applied automatically:
  status IN {"have","partial"}  AND  "Vector" IN spatial_layer_type
```

gis-ui -- web UI

```
gis-ui
# Starts Flask server at http://0.0.0.0:5000
# Open http://localhost:5000 in a browser
```

```
# Requires:  pip install -e ".[server]"
```

End-to-end workflow examples

```
# 1. Preview what is in the database:
set PGPASSWORD=secret
gis-codegen --list-layers

# 2. Save the schema for offline generation:
gis-codegen --save-schema schema.json

# 3. Generate a PyQGIS script with buffer + dissolve:
gis-codegen --platform pyqgis --op buffer --op dissolve -o analysis.py

# 4. Open all layers directly in QGIS:
gis-codegen --platform qgs -o project.qgs

# 5. Load all layers into ArcGIS Pro via a toolbox dialog:
gis-codegen --platform pyt -o loader.pyt

# 6. Generate all catalogue maps as PyQGIS scripts:
gis-catalogue --input catalogue.xlsx --output-dir ./maps/

# 7. Generate a Kepler.gl web map:
gis-codegen --platform kepler -o kepler_map.py
python kepler_map.py

# 8. Use the browser-based web UI:
gis-ui
```

16. Testing

Unit tests (no database or Docker required)

```
# Install dev + server extras:
pip install -e ".[dev,server]"

# Run all unit tests (excludes integration):
python -m pytest tests/ -m "not integration" -v

# With coverage report (must reach 80%):
python -m pytest tests/ -m "not integration" \
    --cov=gis_codegen --cov-report=term-missing

# Run one file:
python -m pytest tests/test_generator.py -v

# Filter by keyword:
python -m pytest tests/ -k "qgs" -v
```

Integration tests (requires Docker)

```
# Install integration extras:
pip install -e ".[dev,integration]"

# Run integration tests (spins up postgis/postgis:15-3.3 container):
python -m pytest tests/test_integration.py -v -m integration

# The tests skip gracefully if testcontainers is not installed.
```

Test suite summary

File	Tests	What is covered
test_generator.py	173	safe_var, pg_type_to_*, _qgs_geom_type, 15 op blocks x 2 platforms, 8 generators (
test_catalogue.py	108	load_catalogue filtering, 5 PyQGIS + 5 ArcPy renderer blocks, symbology dispatch x
test_extractor.py	34	fetch_columns, fetch_primary_keys, fetch_row_count_estimate, extract_schema (mo
test_app.py	11	GET / form rendering, POST /generate happy paths (pyqgis, qgs, pyt), error paths (ba
test_integration.py	19	Live PostGIS container via testcontainers: extract_schema, generate_pyqgis, genera
TOTAL	345	

Unit tests (all except test_integration.py) run in under 2 seconds because the generators are pure string-building functions -- no database connection or GIS library is needed.

CI pipeline (.github/workflows/ci.yml)

- unit job: runs on ubuntu-latest, installs .[dev,server], runs pytest -m 'not integration' with coverage
- integration job: runs on ubuntu-latest (Docker available), installs .[dev,integration], runs pytest test_integration.py -m integration
- Both jobs trigger on push to main/master and on pull requests