

# Exploring the “Unlocking Digital Twins” GitHub repo structure

The Unlocking Digital Twins (UDT) repo is publicly accessible on the Sandtech-EnterpriseAI github account. This is available at the following link: <https://github.com/Sand-EnterpriseAI/udt-clean-water-toolkit>

## Contents of this document

[Introduction to this document](#)

### **[Acronyms required to understand the Project Structure](#)**

[The repository structure](#)

[High level architecture](#)

[Code structure](#)

[CWA](#)

[CWA: cwageodjango](#)

[CWA: examples](#)

[CWA: tests](#)

[CWM](#)

[CWM: cleanwater](#)

[CWM: tests](#)

[Data](#)

[Synthetic data generator](#)

[Data requirements](#)

[Data layers which have been considered from Utilities](#)

[DevOps](#)

[Technology used in module](#)

[Terminology used in module](#)

[Water Industry & Domain Terms](#)

[Water Network Elements & Assets \(from code constants\)](#)

[Workflow Terms](#)

[Partner Organisations](#)

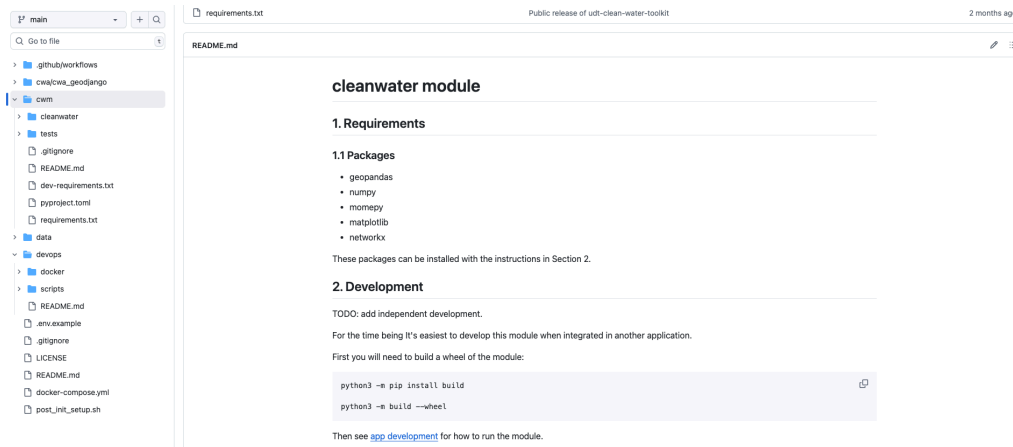
## Introduction to this document

This is a comprehensive explanation of the structure of the code repository and its contents for teams looking to work with, and enhance the code. This is intended for an audience looking to use the repository, and understand where to navigate to find various components. Through this explanation you will find what exists within the repository, how subfolders relate to running code, and whether certain scripts are used in the module, or exist as examples. The required data format (for one's own data to be brought into the repository is touched on). This is not a technical deep-dive into the nuances of the code.

## Acronyms required to understand the Project Structure

- **UDT (Unlocking Digital Twins):** The overarching project name for the Clean Water Toolkit, aiming to facilitate digital twin
- **CWM (Clean Water Module):** The core Python library containing reusable logic for data transformation, network analysis
- **CWA (Clean Water Application):** A Django-based web application that uses the CWM to provide an API and user interface for interacting with the digital twin and m
- **cwageodjango:** The Django/GeoDjango part of the CWA, supporting backend processes and data management, includir

## The repository structure

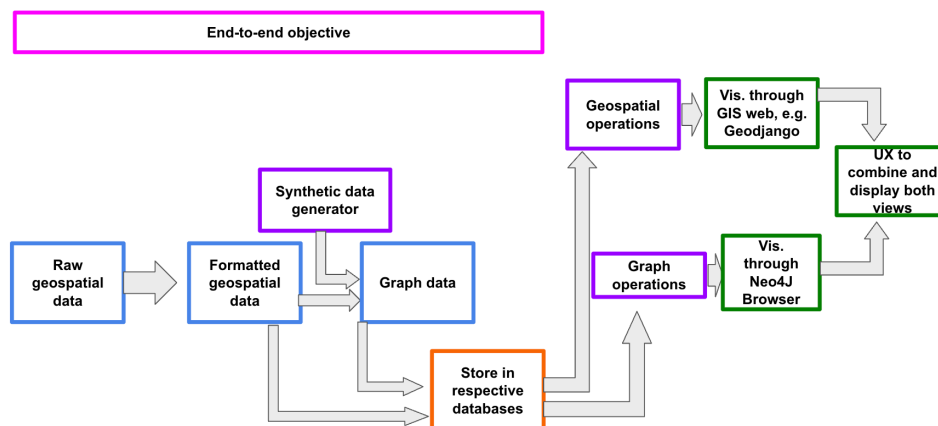


There are 5 areas that are visible in the repository:

- **.github/workflows** - To enforce minimum quality requirements for contributions - every PR must have a description, and the repository must have a non-empty **README.md**.
- **cwa** - The orchestration of the cwm code, to be able to be run through a script as an application. An example of how the module can be used.
- **cwm** - The core module. This is library of code, containing the business logic.
- **data** - The area for data to be stored.
- **devops** - Hold code, scripts, and configuration that support the building, deployment, monitoring, and maintenance of the system.

## High level architecture

Below, we outline the end-to-end objective of the code, including data feeding into it, as well as visualising the insights from the data.

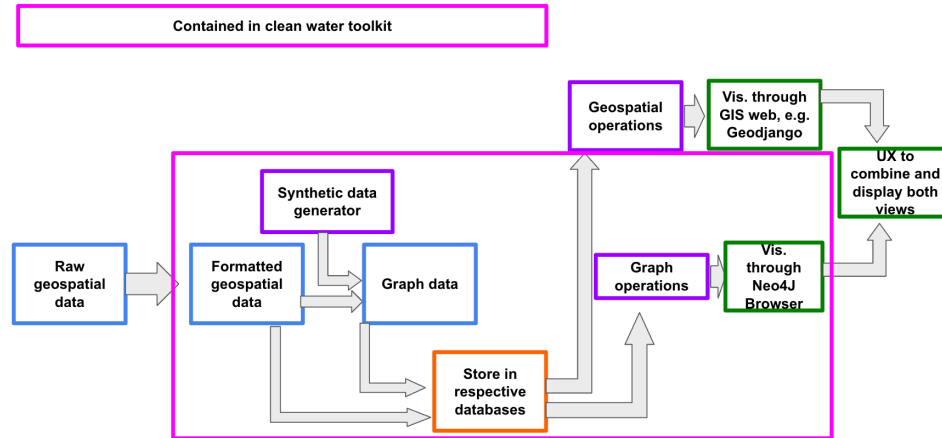


In a data-flow perspective, the CWM will...

- start with geospatial data adhering to required structure; or use the synthetic data generator
- graph data will be generated

- all data is stored in respective databases
- graph data is visualised through **Neo4J Browser**

Therefore, the clean water toolkit includes the components indicated in the pink box below.



Extensions from this (beyond the module)

- geospatial data can be visualised through GIS web, e.g. Geodjango
- a user interface can be created to combine and display both views, and show in a map, geospatial graph, or as a schematic, as needed for the use cases

## Code structure

### CWA

This contains 3 sub-repositories of python files, `cwageodjango` ; `examples` ; `tests` .

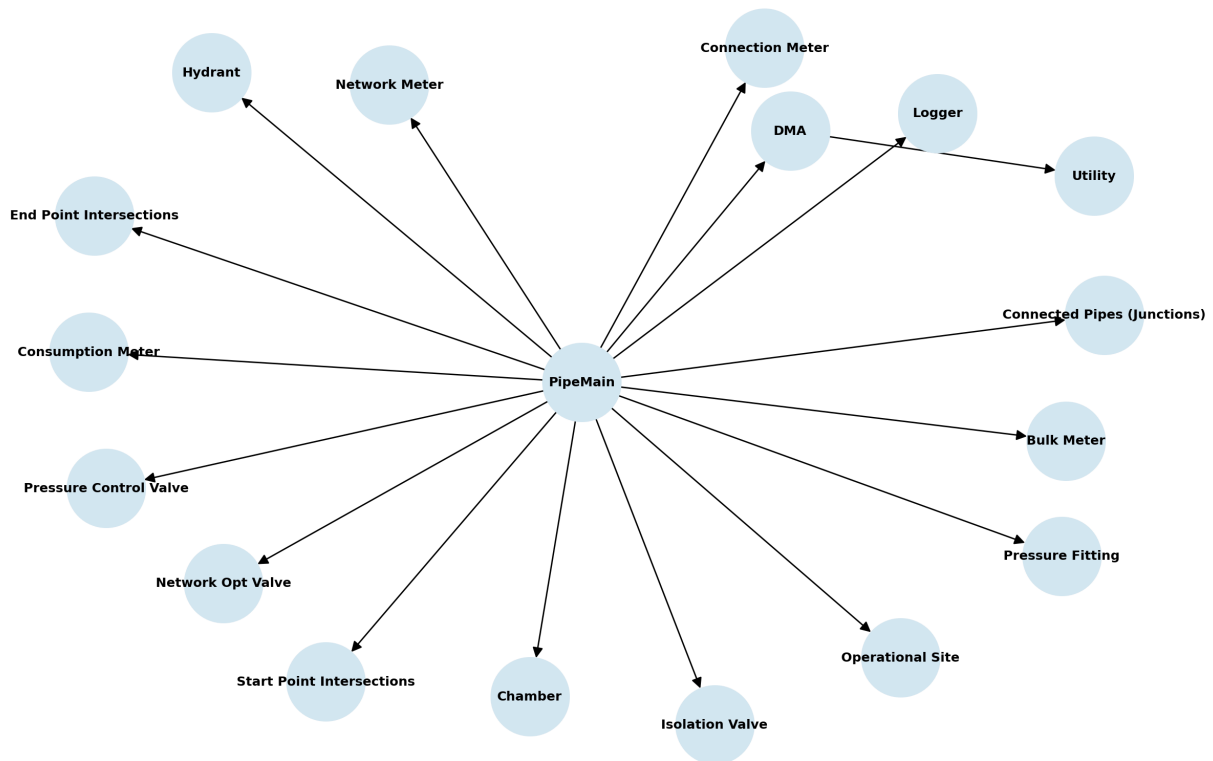
Name	Last commit message	Last commit date
..		
assets	Public release of udt-clean-water-toolkit	2 months ago
config	Public release of udt-clean-water-toolkit	2 months ago
core	Public release of udt-clean-water-toolkit	2 months ago
network	Public release of udt-clean-water-toolkit	2 months ago
utilities	Public release of udt-clean-water-toolkit	2 months ago
waterpipes	Public release of udt-clean-water-toolkit	2 months ago
__init__.py	Public release of udt-clean-water-toolkit	2 months ago

### CWA: cwageodjango

Python and Django files which:

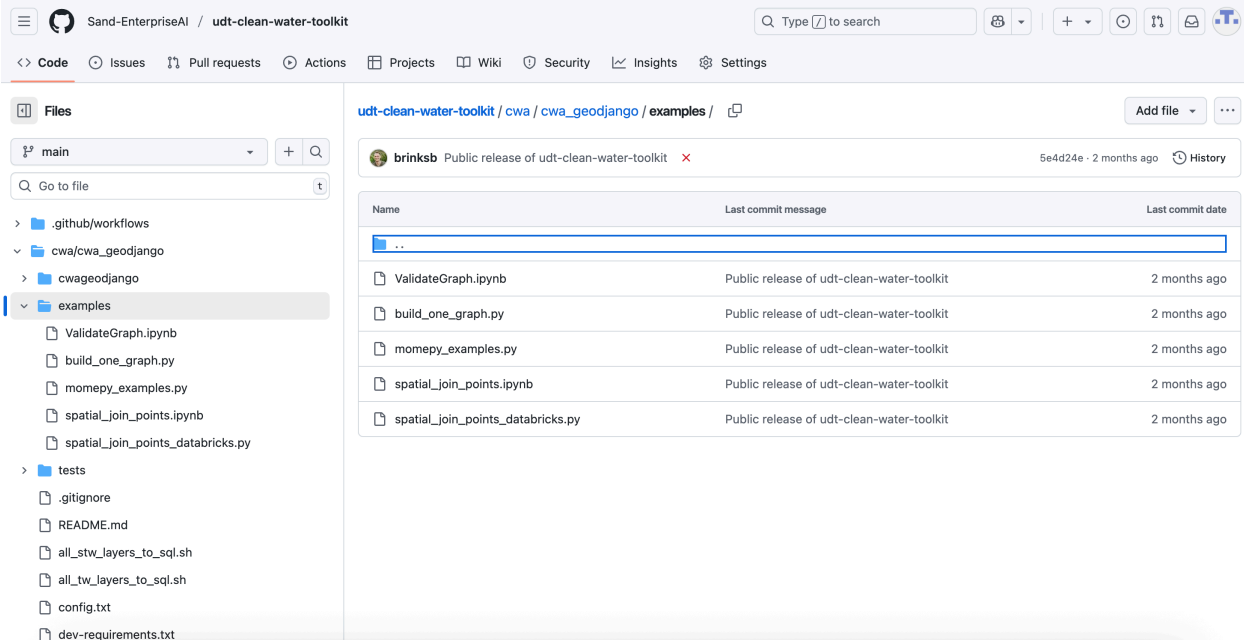
- define a **Django + PostGIS geospatial data access layer** for **pipe mains** in a water network, using a custom `PipeMainsController` class that extends `GeoDjangoDataManager`
- efficiently query, join, filter, and serialize pipe main geometries with related asset and network metadata, returning results as **Django QuerySets, GeoJSON, or GeoDataFrames** for use in APIs or analysis.
- **Django migration script** that sets up the initial database schema for a set of **GeoDjango models** representing water network assets.
- Each utility asset is defined as a python class
- Relationships between assets are defined as properties

**PipeMainsController Relationship Diagram**

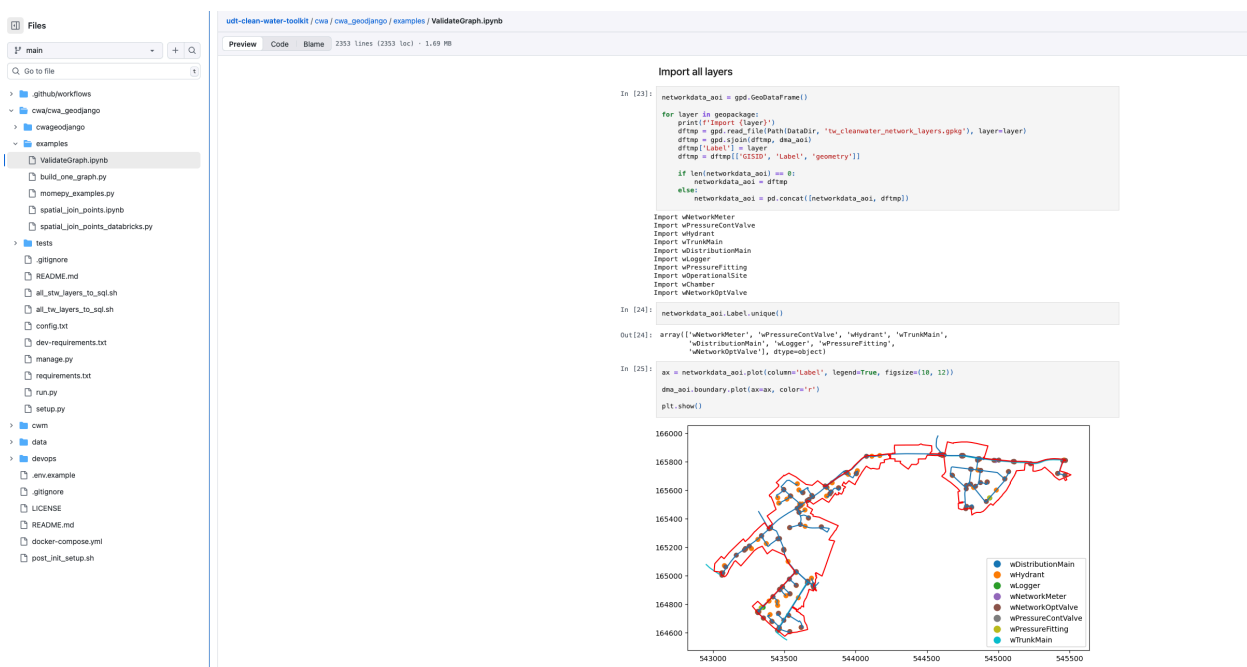


## CWA: examples

`ipynb` notebooks: For a rapid introduction to how graph data, and geospatial data can be manipulated and interacted with - these notebooks will paint a contextualised picture. These notebooks will not run out the box.



- ipython notebooks are illustrative of playgrounds, prior to / surrounding the work in the module
- Illustrated below, are some of the functions of Playing with Neo4j, NetworkX and geopandas - exploring how existing functionality can work



- This notebook takes the geospatial data, analyses, and visualises it using various tools
- Various additional operations, such as geospatial joins are explored - this is a function which is available to work with geospatial data
- **ipynb** notebooks are beyond the scope of the deliverable, but illustrate some of the functionality. We recommend for additional explorations to look at online tutorials.

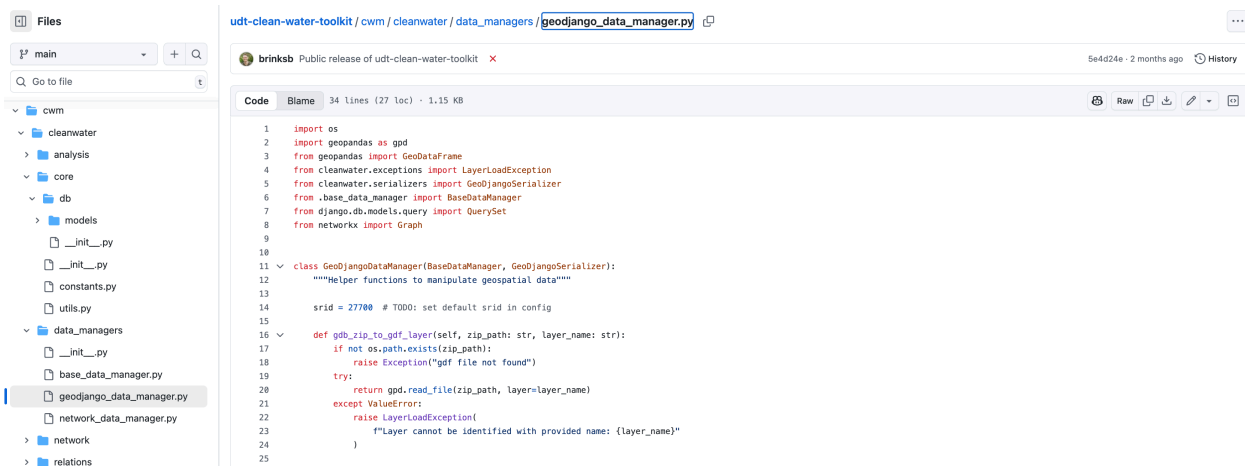
## CWA: tests

- This is empty at the time of publication of the module, and tests can be found within the `CWM/tests` section

## CWM

This contains the core logic. By itself, logic does not do much, it represents facts, constraints and definitions. It needs to be integrated into the application. So this is the logic, the python library, and the CWA is an example of how the module can be used.

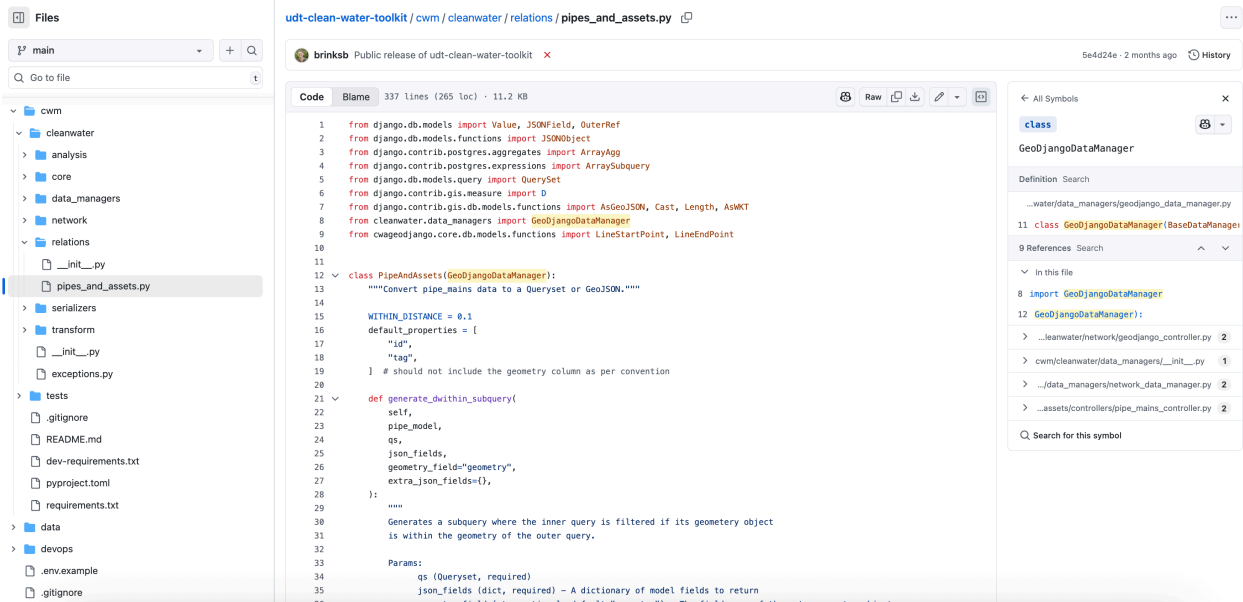
This contains 2 sub-repositories of python files, `cleanwater` ; `tests` .



The screenshot shows a GitHub repository interface for 'udt-clean-water-toolkit'. On the left, a file explorer shows the directory structure: 'cwm' contains 'cleanwater', 'analysis', 'core', 'db', and 'models'. 'cleanwater' contains 'analysis', 'core', 'db', and 'models'. 'db' contains 'models'. 'models' contains 'models'. 'data\_managers' contains 'base\_data\_manager.py', 'geodjango\_data\_manager.py', 'network\_data\_manager.py', 'network', and 'relations'. The main area displays the code for 'geodjango\_data\_manager.py', which is 34 lines long (27 loc) and 1.15 KB. The code includes imports for 'os', 'geopandas', 'GeoDataFrame', 'LayerLoadException', 'GeoJsonSerializer', 'BaseDataManager', 'QuerySet', and 'Graph'. It defines a 'GeoJsonDataManager' class that inherits from 'BaseDataManager' and 'GeoJsonSerializer'. The class has a 'srid' attribute set to 27700. It includes a 'gdb\_zip\_to\_gdf\_layer' method that takes 'zip\_path' and 'layer\_name' as arguments. The method checks if the file exists and raises an exception if not found. It then reads the file and returns a 'GeoDataFrame' object.

## CWM: cleanwater

Functions are contained here for operations within the CWM. This includes helper functions to manipulate data, convert geospatial data, work with geopandas dataframes, create a graph network of assets from a geospatial network of assets, and read data as required.

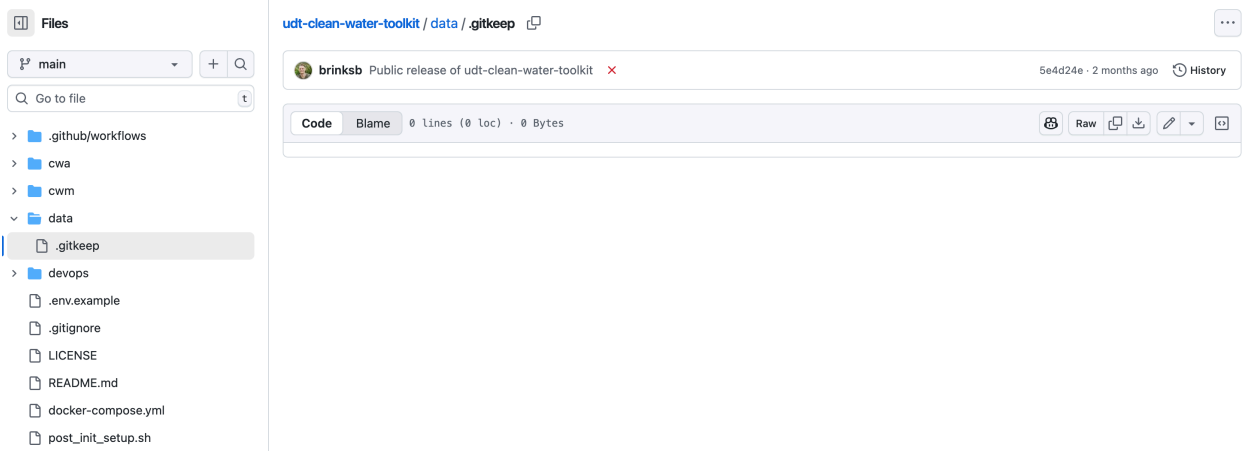


## CWM: tests

- example data - as a python file and CSV
- Space for tests - it is not extensively used within the published module, and there is opportunity to expand this.

## Data

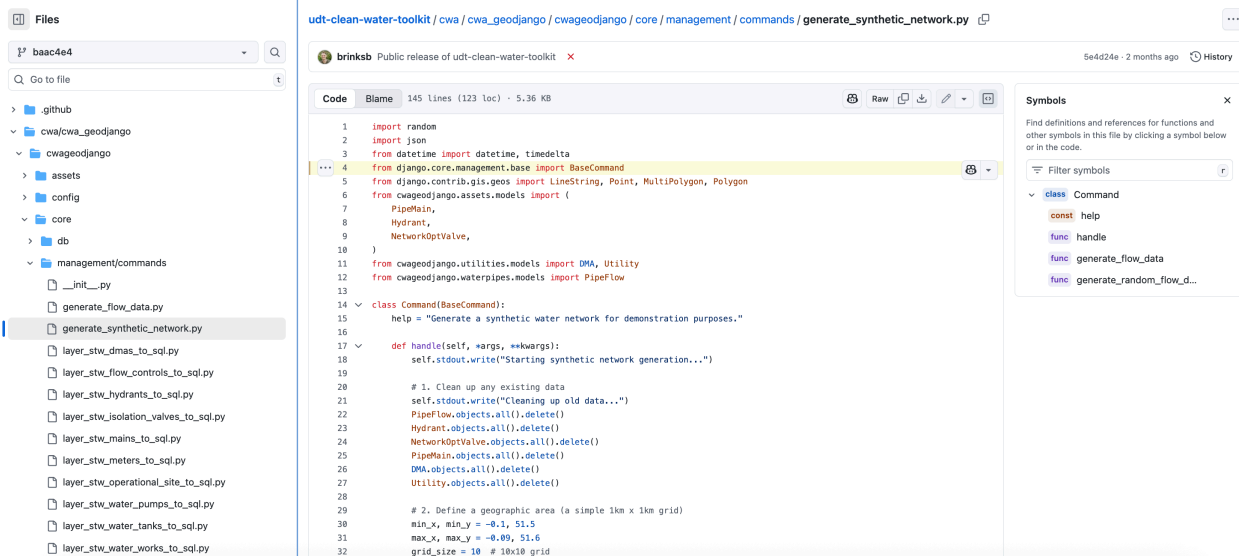
Contains an empty `.gitkeep` file. The “data” folder is referenced, but there is nothing within it



Data added will not automatically be picked up from here, configuration within the code will be needed.

## Synthetic data generator

The data repository is not used with the synthetic data generator, found in the CWA. This instead will populate within the docker environment.



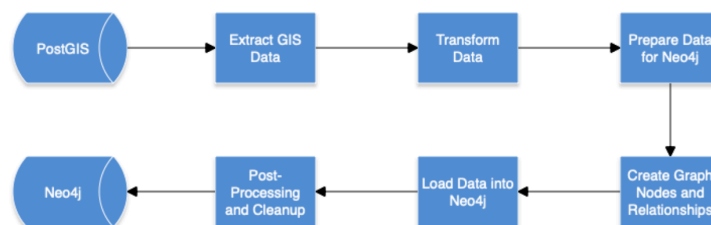
## Data requirements

This is taken directly from the Technical Report, in [Section 4.3](#).

The transformation of GIS data into a graph data structure for water distribution systems involves a series of well-defined steps. This methodology ensures that all network components are accurately represented and allows for efficient querying and analysis. The process leverages pre-validated, cleaned data and focuses on converting spatial data into a graph model suitable for detailed network analysis.

The required structured data includes various network components such as pipes (line features) and assets like pumps, valves, meters, and reservoirs (point features).

The data is loaded into a spatial database in a clean form, eliminating the need for ad hoc data management procedures during transformation.



The figure above illustrates the key steps involved in extracting, transforming, and loading GIS data into the Neo4j graph database. Each step is designed to ensure that the data is accurately represented and that its relationships within the water distribution system are correctly modelled.

### Data preprocessing and validation requirements:

To create a graph data structure that accurately reflects the real-world network, the GIS data must undergo several cleaning, homogenisation, and validation steps:

- **Standardising measurement units** - All measurements, such as pipe diameters, must be standardised across the dataset (e.g., converting all diameters to millimetres). This standardisation ensures uniformity in node and edge properties within the graph, supporting accurate modelling and analysis.



- **Filling missing values** - Any gaps in the data should be addressed by imputing missing values where possible, based on the context provided by subject matter experts or inferred from surrounding data. Complete data ensures that every node and edge in the graph has the necessary attributes for effective network analysis.
- **Removing redundant columns** - Unnecessary columns resulting from multiple generations of data capturing and editing should be removed. Streamlining the dataset helps focus on relevant attributes, simplifying the graph creation process.
- **Ensuring spatial connectivity of pipes** - All connected pipes must be spatially adjacent or "touching," meaning their endpoints coincide without gaps. This check is vital for maintaining correct edge connections between nodes in the graph, accurately reflecting the physical network's layout.
- **Enforcing primary keys** - Every row in the spatial database must have an integer primary key to ensure unique identification and support efficient querying. This enforcement is crucial for maintaining data integrity in the graph database.
- **Maintaining unique identifiers ('Tag' Column)** - Each row should contain a unique string identifier ('tag') to prevent duplication and to enable easy reference to specific entries. Unique identifiers ensure that each node and edge in the graph is distinct, avoiding confusion and errors.
- **Consolidating pipe data** - All pipe-related information should be stored in a single table to simplify data management. This consolidation ensures that pipes are consistently represented as edges in the graph structure.
- **Enforcing line geometry for pipes** - Pipes must be represented as line geometries, not multiline geometries, to model each pipe as a single continuous line. This representation aligns with the graph's requirement for clear and precise edges.
- **Preventing self-intersection of pipes** - Pipes should not intersect with themselves, which could lead to inaccuracies in spatial analysis. In graph terms, this prevents edges from looping back, preserving the network model's integrity.
- **Representing assets as point geometries** - All assets must be represented as points to ensure accurate location mapping and easy analysis in relation to other spatial data. This step ensures assets are correctly mapped as nodes within the graph.
- **Proximity-based node attachment for assets** - Point assets within a defined distance (suggested default of 1 cm) of each other are considered to be at the same spatial position and are attached to the same node in the graph structure. This ensures that the graph model accurately reflects real-world connections and prevents redundant nodes.

By combining these cleaning, homogenisation, and validation steps, GIS data is thoroughly prepared for transformation into a graph data structure, ensuring accurate representation of the water distribution system for reliable analysis and decision-making.

**Section 4.4** in the technical report outlines the processes required to be followed to retrieve and explode line segments; and catalogue dictionaries and relationships.

## Data layers which have been considered from Utilities

This is taken directly from the Technical Report, in **Table 6.1**.

**Table 6.1:** List of assets used in building the model for Severn Trent Water and Thames Water

Severn Trent Water	Thames Water
Trunk Pipes	Trunk Main
Distribution Pipes	Distribution Main
Flow control	Connection main
Hydrants	Chamber
Meter	Connection meter
Water Pumps	Consumption meter
Water Tanks	Hydrant
Water Works	Logger
	Network meter
	Network operational valve
	Operational Site
	Pressure control valve
	Pressure Fitting

## DevOps

The devops folder orchestrates the application, and integrated the module into the application, as well as the surrounding services.

The screenshot shows the GitHub interface for the repository 'udt-clean-water-toolkit'. On the left, the 'Files' sidebar shows the directory structure, with the 'devops' folder selected. The main area displays the commit history for the 'devops' folder. The table below represents the data shown in the commit history:

Name	Last commit message	Last commit date
..		
docker	Public release of udt-clean-water-toolkit	2 months ago
scripts	Public release of udt-clean-water-toolkit	2 months ago
README.md	Public release of udt-clean-water-toolkit	2 months ago

**Docker:** - <https://www.docker.com/>, to...

- Spin up new environments quickly
- Integrate with your existing tools
- Containerize applications for consistency
- Includes configuration requirements to spin up images as needed of data storage services
- Includes configuration requirements to spin up images as needed of data processing services

**Scripts:**

- `change_multi_geometry_to_single_geometry.py` - This Python script is a geospatial data transformation and enrichment tool built to take an old GeoPackage of water network layers, clean and transform the geometry, enrich it with additional attributes, and save a new cleaned GeoPackage.
- `convert-vector-format.sh` - This .sh file is a geospatial data extraction, transformation, and loading (ETL) script for converting ESRI File Geodatabase (FGDB) layers into a GeoPackage (GPKG) (or other formats, if modified).
- `data_audit.py` - Geospatial Database Audit Script - it inspects an ESRI File Geodatabase (.gdb) and generates a text-based report of its layers, geometry, projection, network membership, and column-level completeness.
- `seven_trend_data_conversion.sh` - This .sh file is another geospatial data conversion and enrichment script, it processes Shapefile (.shp) directories and combines them into a GeoPackage (.gpkg).

## Technology used in module

- **GeoDjango**: A Django framework extension for building GIS (Geographic Information System) web applications.
- **PostGIS**: An extension to PostgreSQL that adds support for geographic objects, allowing spatial queries.
- **Neo4j**: A graph database used for representing and analyzing water networks as graphs (nodes and edges).
- **NetworkX**: A Python library for creating, manipulating, and studying complex networks/graphs.
- **GeoPandas**: A Python library for working with geospatial data.
- **Momepy**: Python tool for urban morphology analysis.
- **WNTR**: Water Network Tool for Resilience, a Python package for simulating and analyzing water distribution networks.

## Terminology used in module

### Water Industry & Domain Terms

- **Digital Twin**: A digital replica of a physical system (here, a water network), used for simulation, analysis, and planning.
- **Geospatial Model**: Data/model based on spatial (location-based) information, often used in GIS.
- **Graph Technology**: Use of nodes and edges to model relationships, enabling advanced analytics like pathfinding and ce

### Water Network Elements & Assets (from code constants)

- **Pipe Main**: The principal pipe in a water distribution network.
- **Pipe End / Pipe Junction**: Nodes representing the endpoints or junctions of pipes.
- **Point Asset**: General term for any discrete (point-like) asset in the network.
- **Hydrant**: An outlet for drawing water, typically for firefighting.
- **Logger**: Device that records data such as flow or pressure.
- **Meter (Network, Connection, Consumption, Bulk, Revenue)**: Devices measuring water flow at various points and for di
- **Operational Site**: A facility or site in the operational scope of the water network.
- **Pressure Control Valve / Pressure Fitting**: Devices for managing and monitoring water pressure.
- **Isolation Valve**: Valve used to isolate a section of the network.
- **Regulator**: Device that regulates flow or pressure.
- **Non-Return Valve**: Prevents water from flowing backward.
- **Water Pump / Water Pumping Facility**: Devices and facilities for moving water through the network.
- **Water Tank**: Storage tank for water.
- **Water Work / Water Treatment Work**: Facilities for water treatment or processing.

- **Potable Water Storage:** Storage for drinkable water.
- **DMA (District Metered Area):** A discrete area of the water network with metered input, used for monitoring and managing

### **Workflow Terms**

- **Ingestion:** The process of importing external data (e.g., from GeoPackages, CSVs) into the database.
- **Synthetic Flow Data:** Artificially generated data to simulate water flow for testing or analysis.
- **SRID (Spatial Reference System Identifier):** A unique ID used to reference a spatial coordinate system.

### **Partner Organisations**

- **Ofwat:** The Water Services Regulation Authority, a regulator for the water sector in England and Wales.
- **Thames Water:** Major UK water utility.
- **Severn Trent Water:** Major UK water utility.