Backend

Transforming the bounding box

Because the bounding box for the study is given in RD new and the overture query needs WGS84 coordinates, I wrote a function to transform the input bounding box using the ```Transformer``` from the pyproj library.

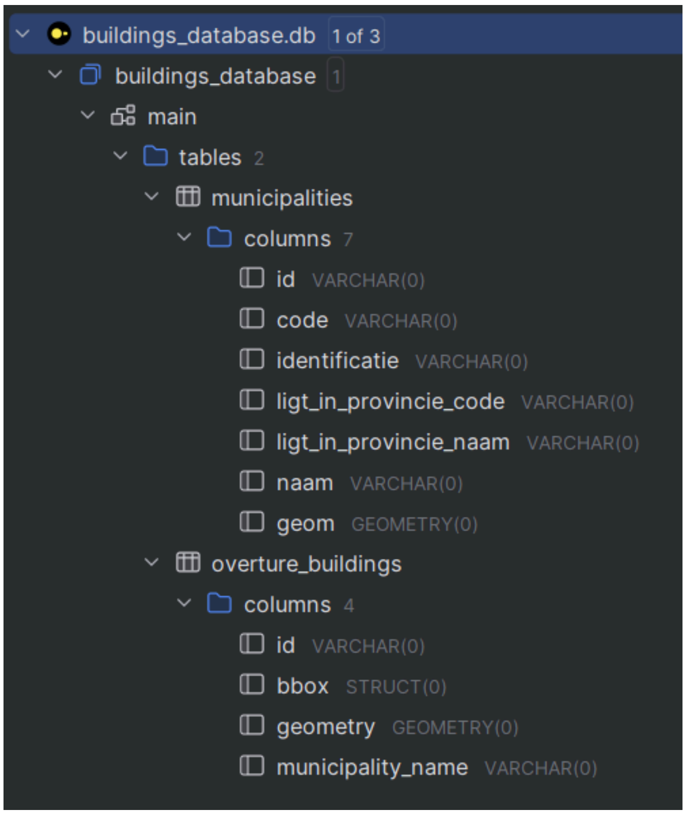
Getting the data

Getting the Overture and PDOK data was a bit tedious, but after really carefully following the examples provided in the docs, I got it working. The most confusing part to me was getting the correct link for the Overture data. I ended up going with s3://overturemaps-us-west-2/release/2025-10-22.0/theme=buildings/type=building/*.parque, which was in one of the examples and seemed to return a reasonable number of buildings that matched what I expected when loaded into QGIS.

Setting up the DuckDB database was actually really pleasant. After using POSTGIS, not having to bother with something like pgAdmin was nice.

Results

Initially, I wrote a small print function to print my database to the terminal, but I mostly used the integrated database viewer within PyCharm, which worked quite nicely. The image below shows the overview that was really useful for checking my tables and columns. As you can see, I chose to remove most attributes from the Overture data because they weren't needed for the API I was going to build and would clutter my database with many nearly empty columns.



```
buildings_database.db  1 of 3
  buildings_database  1
    main
      tables  2
        municipalities
          columns  7
            id  VARCHAR(0)
            code  VARCHAR(0)
            identificatie  VARCHAR(0)
            ligt_in_provincie_code  VARCHAR(0)
            ligt_in_provincie_naam  VARCHAR(0)
            naam  VARCHAR(0)
            geom  GEOMETRY(0)
        overture_buildings
          columns  4
            id  VARCHAR(0)
            bbox  STRUCT(0)
            geometry  GEOMETRY(0)
            municipality_name  VARCHAR(0)
```

API

For the API implementation, I used the FastAPI and Pydantic libraries after watching the introductory course by freeCodeCamp.org. Once I got the first endpoint working, implementing the others wasn't so bad. I was planning on learning this framework either way, so doing it for this course is a nice way to stop procrastinating it. I tried to set up a clean file using Pydantic schemas, sensible defaults, and error handling.

Pagination

- GET /collections/{municipality}/items and GET /buildings/bbox implement pagination because they return potentially thousands of building features.
- GET /collections does not because this endpoint lists the municipalities (Collections), not the buildings (Features). Since there are only a few dozen municipalities, sending them all at once is fine.
- GET /collections/{municipality}/items/{building_id} Also does not because this endpoint fetches a single building by definition.

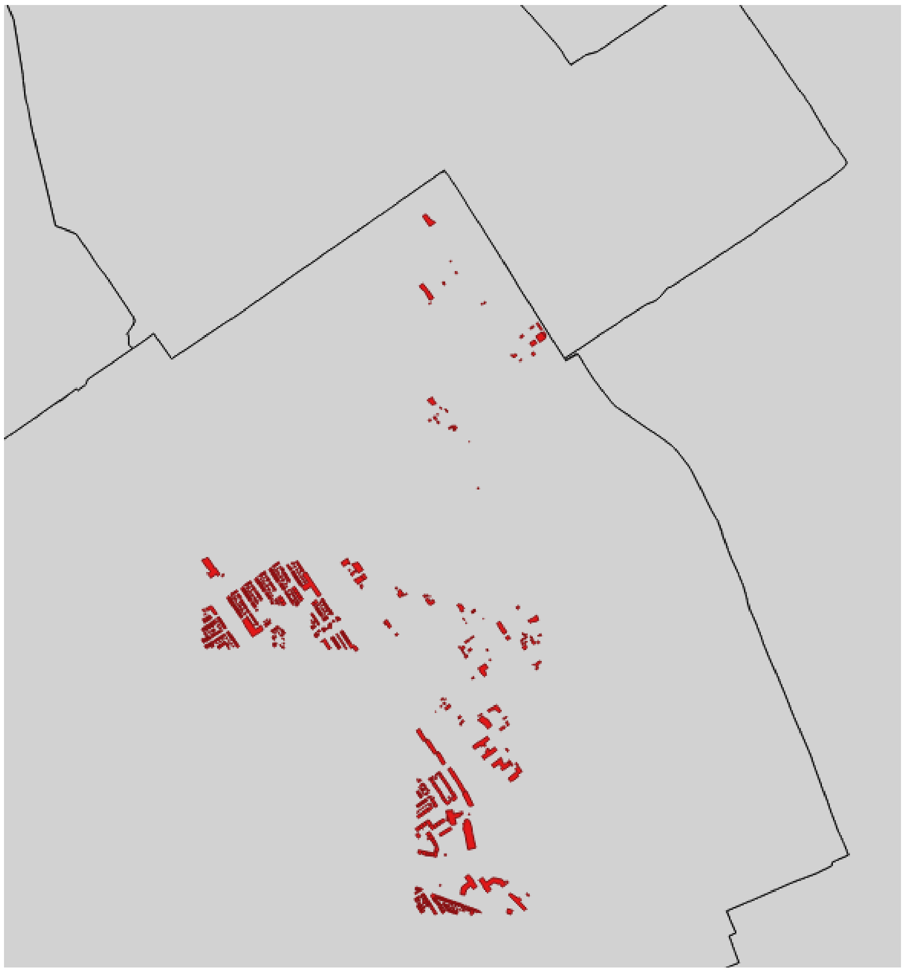
I also spent more time than I would have liked on getting the next and previous links working. Especially the string formatting and, for example, now showing a previous link when the offset is 0 was a hassle, but ended up working in the end.

Result

To test the API, I used the built-in/docs endpoint to run test calls and make sure everything was working as expected. The code below shows the start of the response for http://127.0.0.1:8000/collections/Delft/items. The response shown the structure as laid out in the assignment is implemented and extended with pagination (numberReturned, limit, offset, links). Note that there are no pagination parameters included, so the defaults get used. Also, there is no previous link since it is the first chunk.

```
{
  "type": "FeatureCollection",
  "numberMatched": 27676,
  "numberReturned": 50,
  "limit": 50,
  "offset": 0,
  "links": [
    {
      "href": "http://127.0.0.1:8000/collections/Delft/items",
      "rel": "self",
      "type": "application/geo+json",
      "title": "Current page"
    },
    {
      "href": "http://127.0.0.1:8000/collections/Delft/items?offset=50&limit=50",
      "rel": "next",
      "type": "application/geo+json",
      "title": "Next page"
    }
  ],
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [
            [85395.3742427547, 449672.496138249],
            etc...
          ]
        ]
      },
      "properties": {
        "id": "58caa22a-abc9-4f1c-b822-fb181f8c5523",
        "municipality_name": "Delft"
      }
    },
```

To check the response, I requested the first 1000 buildings in Delft. The image below shows the GeoJSON response loaded into QGIS.



Questions

The /buildings/bbox endpoint is non-compliant because OGC wants all feature access, including spatial filtering, to happen through the unified /collections/{municipality}/items path using a bbox query parameter ([OGC API - Features - Part 3: Filtering](#)). To become OGC compliant, the /buildings/bbox endpoint should be removed, and the logic for filtering with a bounding box should be integrated in the /collections/{municipality}/items endpoint.