CS162 Solo Project Weekly Report: Week 06

CS162, Intro. to Computer Science II
Semester 2, AY2023/24

Solo Project

Full Name	Student ID
NGUYEN Hoang The Kiet	23125023

Lecturer: DINH Ba Tien (PhD)
TAs: HO Tuan Thanh (MSc), NGUYEN Le Hoang Dung (MSc)

Ho Chi Minh City, March 31, 2024

Tasks by Week

Week 06

1. (Chapter 1) PROJ: Coordinate Transformation
2. (Chapter 2) GeoJSON: Geosptial Data Format
3. (Section 6.3) Path (the `path` module), (Subsection ©) `PathQuery`
4. (Chapter 3) Shapely: Spatial Analysis
5. (Chapter 4) R-tree: Data Structure for Spatial Data
6. (Chapter 5) LangChain: Apply Large Language Model to Querying Databases

Contents

List of Figures

List of Listings

Chapter 1

PROJ: Coordinate Transformation

PROJ[2] is a generic coordinate transformation software that transforms geospatial coordinates from one *coordinate reference system (CRS)* to another.

- **Cartographic projections**. Mathematical transformations that project the Earth, represented by a uniform sphere (or a spherical ellipsoid) into the 2D Cartesian coordinate system.
- **Geodetic transformations**. Mathematical transformations from one CRS to another.

`pyproj`[3] is the Python interface to PROJ. The package can be installed like any other Python libraries (via `pip`, `conda`, etc.)

```
pip install pyproj
```

Listing 1.1: PROJ: Installing `pyproj`

1.1 | Some common Coordinate Reference System (CRS)

(a) | EPSG:4326 / WGS-84: The World Geodetic System 1984

WGS-84 is a reference coordinate system that consists of

- **A reference ellipsoid**. WGS-84 models the Earth as a **spherical ellipsoid** that approximates the Earth's **geoid**, with parameters:
  - *Semi-major axis* $a = 6\ 378\ 137.000$m
  - *Flattening* $f = 1 : 298\ 257\ 223\ 563$
- **A coordinate system**. WGS-84 uses **longtitudes** (λ) and **latitudes** (φ) to position a point on the globe. When the distance of a point to the Earth's surface is considered, **altitudes** (h) is also introduced.

WGS-84 is the best geodetic system up to date. The system has an absolute accuracy of $1 - 2$ metres. Hence, many geodetic applications rely on the WGS84.

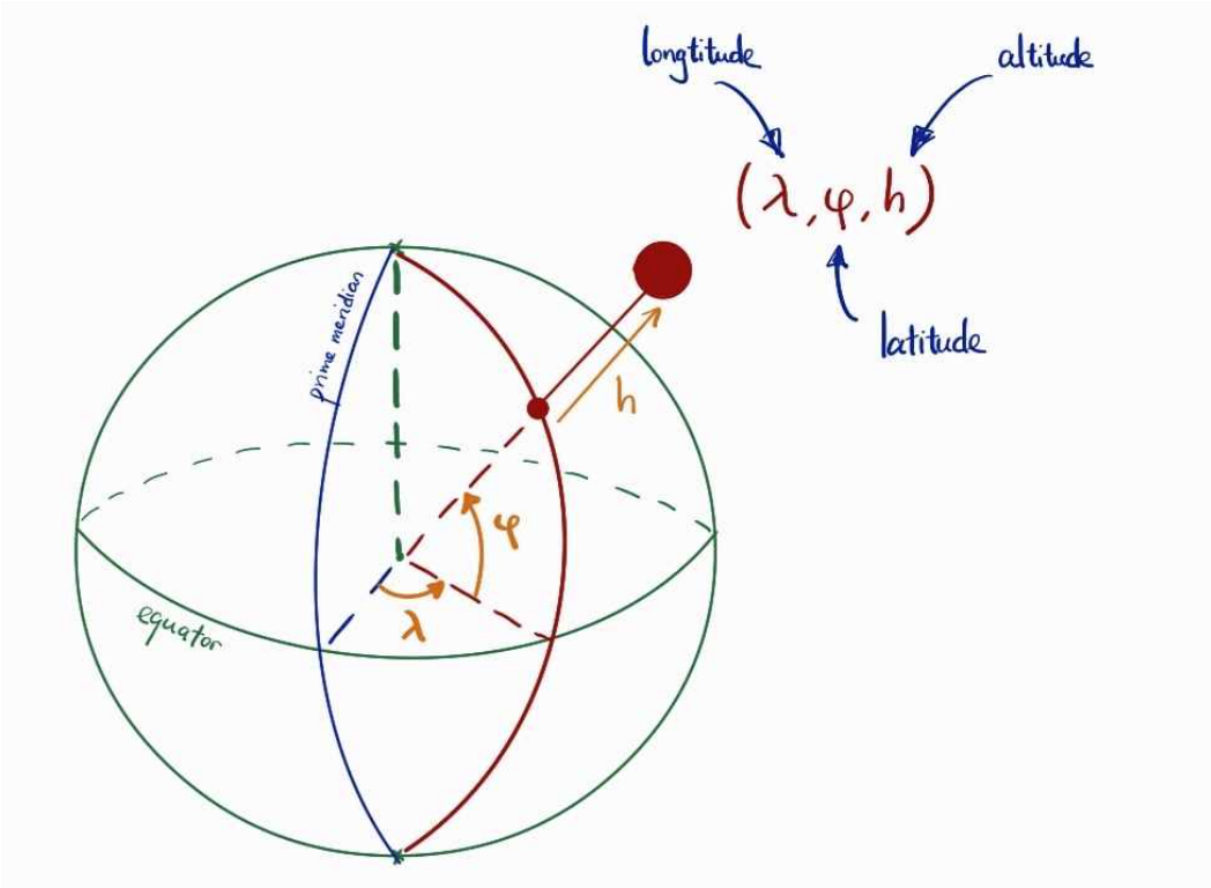


Figure 1.1: PROJ: The WGS-84 ellipsoid model and coordinate system

The WGS-84 standard is currently being used in the Global Positioning System (GPS), and its model of the Earth is adopted by many countries' CRS, including Vietnam's VN-2000.

ⓑ | EPSG:3405 / VN-2000: Vietnam's National coordinate system 2000

Vietnam is currently using the VN2000 coordinates system, first established by *Decision No. 83/2000/QD-TTg on the use of Vietnam's National references and Coordinates Systems*[4]. Prior to VN2000, Vietnam had used the HN-72 system, a system that was commonly used in former socialist countries.

- **Reference ellipsoid**. VN-2000 shares its reference ellipsoid with WGS-84.
- **Coordinate system**. According to the international UTM Confirmed Cylindrical Transverse Mercator Projection. Vietnam uses the UTM projection on Zone 48N, which projects Vietnam onto a flat surface equipped with a Cartesian coordinate system, with minimum distortions.

Each CRS has their own advantages. Whilst WGS-84 excels in locating positions and is more common in GPS systems, tasks like calculating distances, finding shortest route, etc. require a flat coordinate system. Projected CRS such as VN-2000 is, therefore, a better-fit.

	WGS-84 (EPSG:4326)		**VN-2000** (EPSG:3405)	
	lat (λ)	**lng** (φ)	x	y
min	102.14	8.33	184767.75	999099.0
centre	-/-	-/-	589204.99	1754291.1
max	109.53	23.4	920895.91	2595190.43

Table 1.1.1: Coordinate bounds in WGS-84 and VN-2000

1.2 | Converting from one CRS to another

ⓐ | `Transformer.from_crs`

This is the `Transformer` class constructor. It takes two main arguments:

- `crs_from: pyproj.crs.CRS | str`: source CRS
- `crs_to: pyproj.crs.CRS | str`: desired CRS after transformation

ⓑ | `Transformer.transform`

This function transform point(s) from the `crs_from` CRS to the `crs_to` CRS. The main parameters are

- `xx`: input x coordinate(s)
- `yy`: input y coordinate(s)
- `zz`: input z coordinate(s) (optional)
- `tt`: input t coordinate(s) (optional)
- `radians`: Input t coordinates (optional)
- `errcheck`: if `True`, raises errors; otherwise, returns `inf`
- `inplace`: if `True`, attempts to write the results onto the input array instead of returning a new array.

The function supports transformation in at least 2 dimensions and at most 4 dimensions.

```
class pyproj.transformer.Transformer:
    # ...
    @staticmethod
    def from_crs(crs_from: Any, crs_to: Any, **kwargs)
        -> Transformer:
        """
        Get all possible transformations from a :obj:`pyproj.crs.CRS`
        or input used to create one.
        ...
        """

    def transform(
        self,
        x: Any,
        y: Any,
        z: Any = None,
        t: Any = None,
        radians: bool = False,
        errcheck: bool = False,
        inplace: bool = False,
        **kwargs
    ) -> tuple[Any, Any] | tuple[Any, Any, Any] | tuple[Any, Any, Any, Any]:
        """
        Transform points between two coordinate systems.
        ...
        """

    # ...
```

Listing 1.1: PROJ: Implementation in `pyproj.Transformer` class. [1]

1.3 | Example: Converting from WGS-84 to VN-2000

```python
import pyproj

wgs84_vn2000_tf = pyproj.Transformer.from_crs('epsg:4326', 'epsg:3405')
vn2000_wgs84_tf = pyproj.Transformer.from_crs('epsg:3405', 'epsg:4326')

def wgs84_to_vn2000(lat: float, lng: float) -> tuple[float, float]:
    return wgs84_vn2000_tf.transform(lat, lng)

def vn2000_to_wgs84(x: float, y: float) -> tuple[float, float]:
    return vn2000_wgs84_tf.transform(x, y)
```

Listing 1.1: PROJ: Converting `(lat, lng)` in WGS-84 to `(x, y)` in VN-2000. Module defined in `helper/crs_convert.py`

Chapter 2

GeoJSON: Geosptial Data Format

GeoJSON is a format for encoding a variety of geographic data structures [5]. It is an extension of the JSON format, defined by the RFC7946 standard [6].

2.1 | GeoJSON Viewers

ⓐ | geojson.io

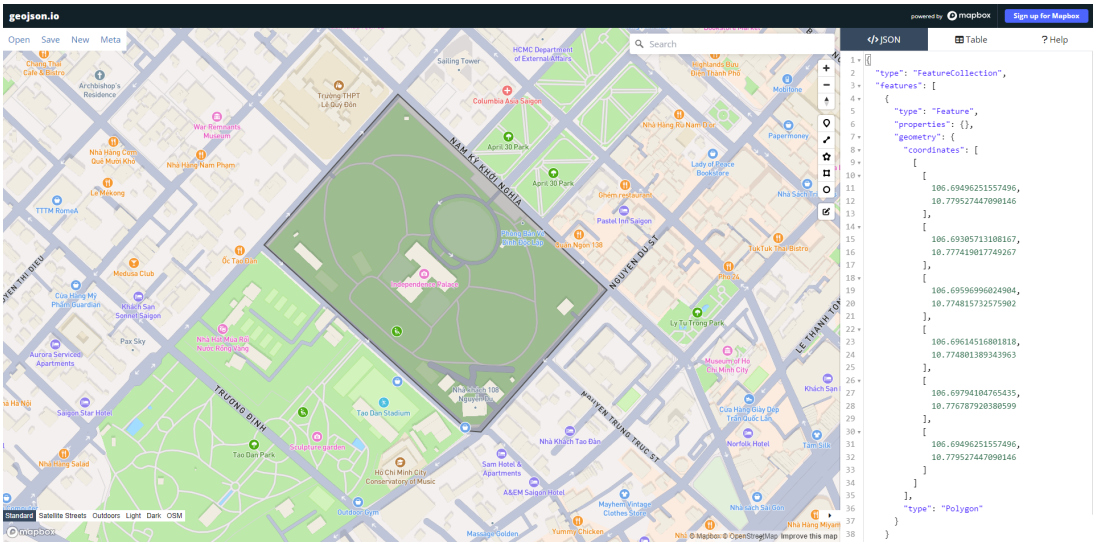


Figure 2.1: GeoJSON: *geojson.io*

geojson.io is a website designed for working with GeoJSON files. It supports:

- **Creating and Editing GeoJSON data**. GeoJSON objects can be editted manually or via graphical tools on the web.
- **Interchanging geodata between different formats**. geojson.io supports importing from and exporting to various formats, including *.geojson, *topojson, *.csv, *.kml, etc.
- **Visualising data on maps**. GeoJSON objects are overlayed on real maps such as satellite maps and street maps.

(b) | Geo Data Viewer (a Visual Studio Code snippet)

This is a Visual Studio Code extension that helps visualing local GeoJSON files. To use the extension, simply install via VSCode Marketplace, then at the .geojson file, choose the *Geo: View Map* option from the *Command Pallete*.

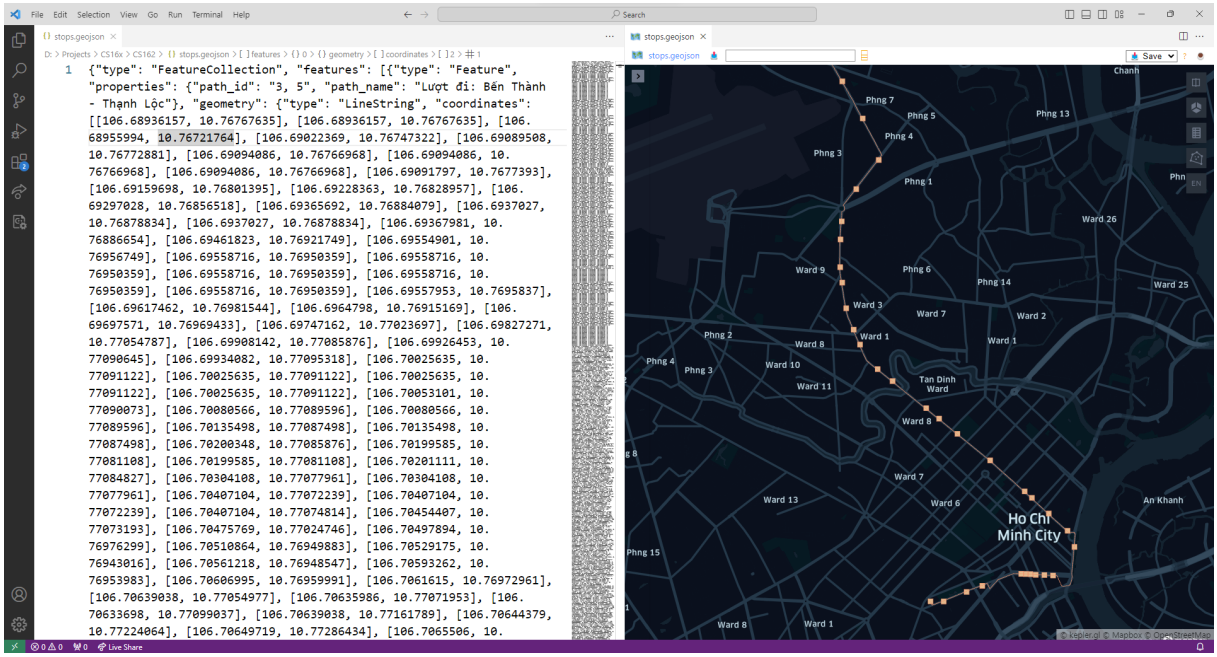


Figure 2.2: GeoJSON:
GeoJSON file in plain *(left)*, Visualisation in *Geo Data Viewer (right)*

2.2 | GeoJSON Features

ⓐ | Feature

A Feature object represents a spatially bounded thing.

- **Type**. Every Feature object has a `"type"` attribute with value `"Feature"`.
- **Geometry**. Defines the shape of the object and its coordinates.
- **Properties**. Properties are optional and do not affect the shape of the object, but may come in handy when objects are visualised on the map.

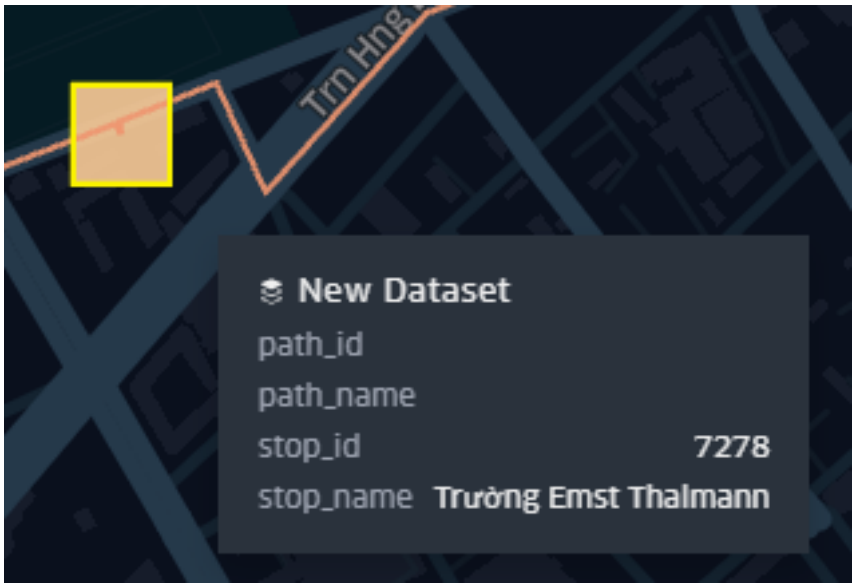


Figure 2.3: GeoJSON: Properties of a polygon show up when hovered

ⓑ | GeometryCollection

A GeometryCollection object contains a list of `"geometries"`, each of which is a Geometry object (See GeoJSON Objects).

ⓒ | FeatureCollection

A FeatureCollection object contains a list of `"features"`, each of which is a Feature object. This is the highest object in the hierarchy of the GeoJSON object tree.

```geojson
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
          "flight": "SGN - SIN",
          "time": "2:05",
          "cost": 4959000
      },
      "geometry": {
        "coordinates": [
            [106.7179127, 10.8085596], [103.7174418, 1.4671041]
        ],
        "type": "LineString"
      }
    },
    {
      "type": "Feature",
      "properties": {
        "Airport": "Tan Son Nhat"
      },
      "geometry": {
        "coordinates": [106.7168838, 10.7699593],
        "type": "Point"
      }
    },
    {
      "type": "Feature",
      "properties": {
        "Airport": "Changi"
      },
      "geometry": {
        "coordinates": [103.6990521, 1.3358573],
        "type": "Point"
      }
    }
  ]
}
```

Listing 2.1: GeoJSON: Example of a complete .geojson file

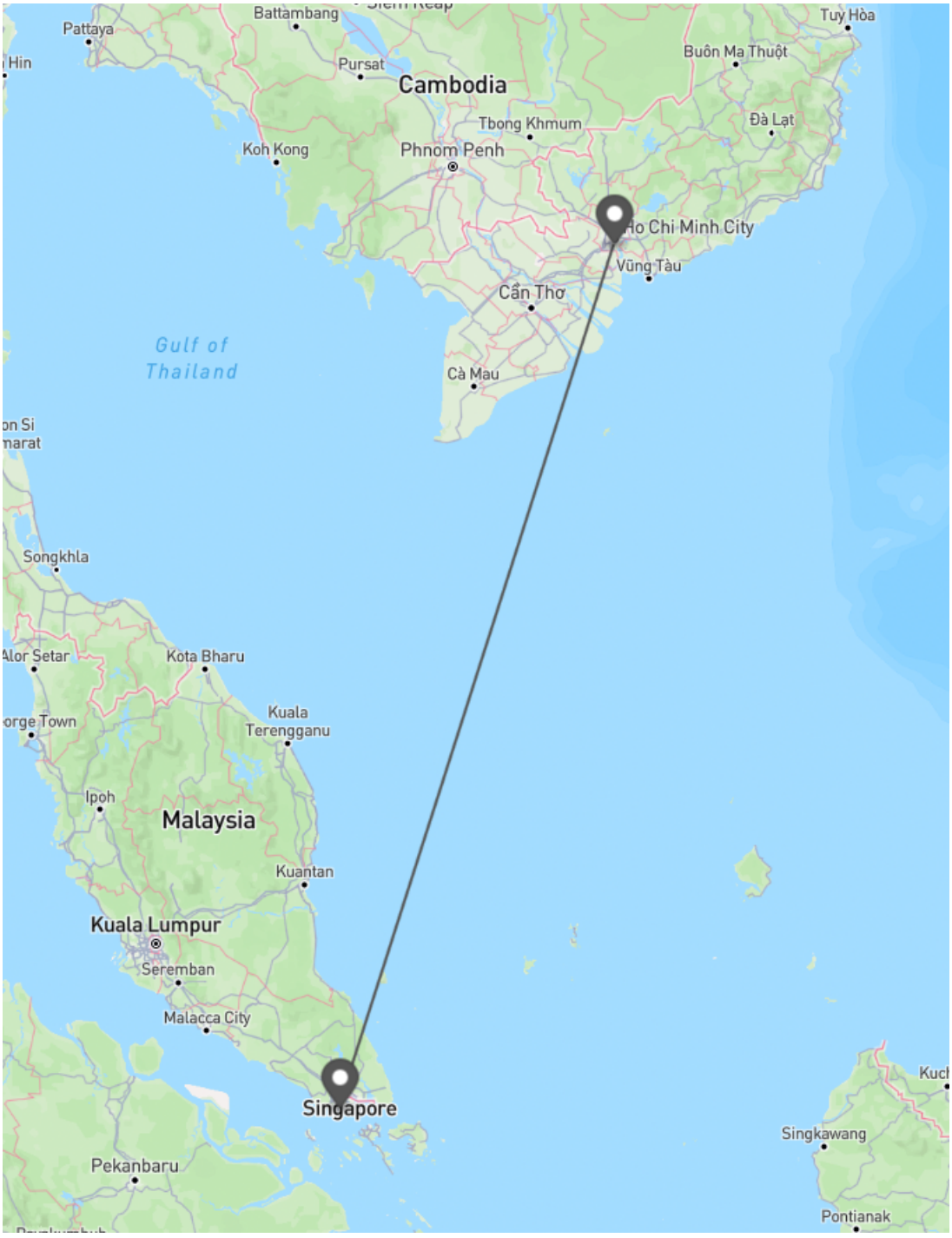


Figure 2.4: GeoJSON: Visualisation on satellite map

2.3 | GeoJSON Objects

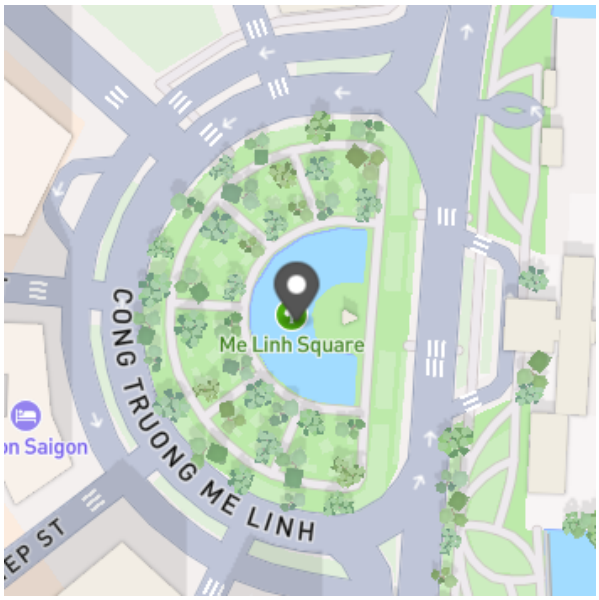
All GeoJSON gemetry objects have some common properties:

- **Type**. can be one of the geometry types: `Point`, `MultiPoint`, `LineString`, `MultiLineString`, and `Polygon`, `MultiPolygon`.
- **Coordinates**. A list of coordinates that defines the object. GeoJSON uses the WGS-84 coordinate system (latitude, longtitude, elevation), where elevation is optional.

  These basic geometry objects build up what is called a Spatial Data Model, which will be discussed in Section 3.1 when we dig into spatial data analysis and manipulation.

ⓐ | Point

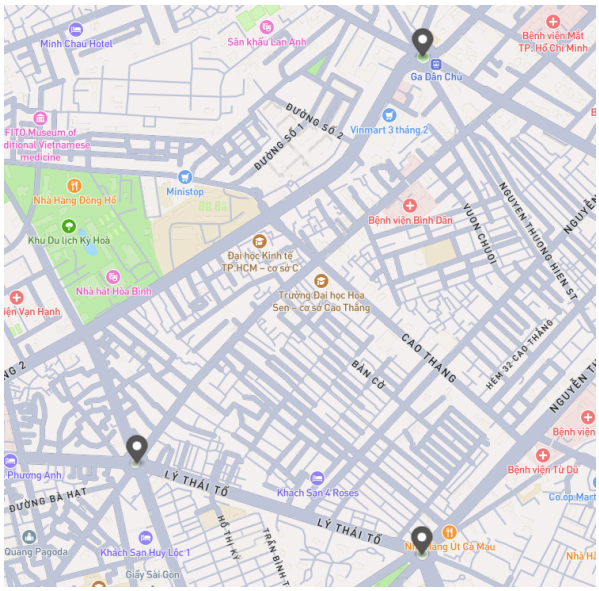
```
{
  "type": "Feature",
  "properties": {
    "Name": "Me Linh Square"
  },
  "geometry": {
    "coordinates": [
        106.7063463, 10.7752786
    ],
    "type": "Point"
  }
}
```



Listing 2.1: GeoJSON Point

ⓑ | MultiPoint

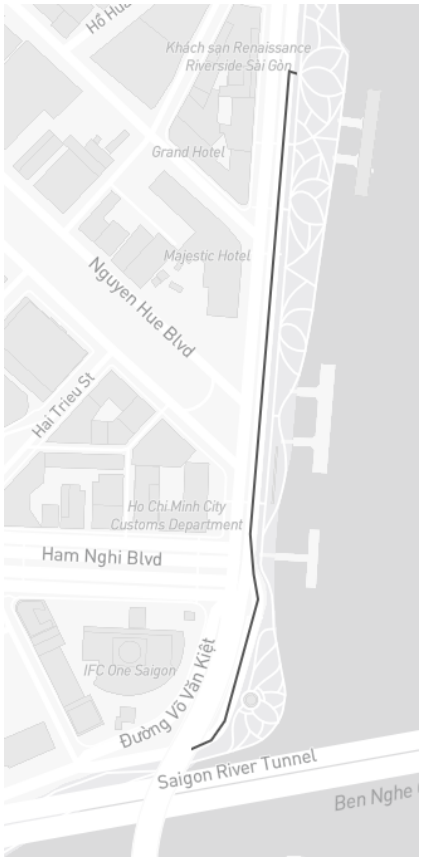
```
{
  "type": "Feature",
  "properties": {
    "Name": "Roundabouts"
  },
  "geometry": {
    "coordinates": [
      [106.6816905, 10.7778276],
      [106.6816538, 10.7654209],
      [106.6744136, 10.7677078]
    ],
    "type": "MultiPoint"
  }
}
```



Listing 2.2: GeoJSON MultiPoint

ⓒ | LineString

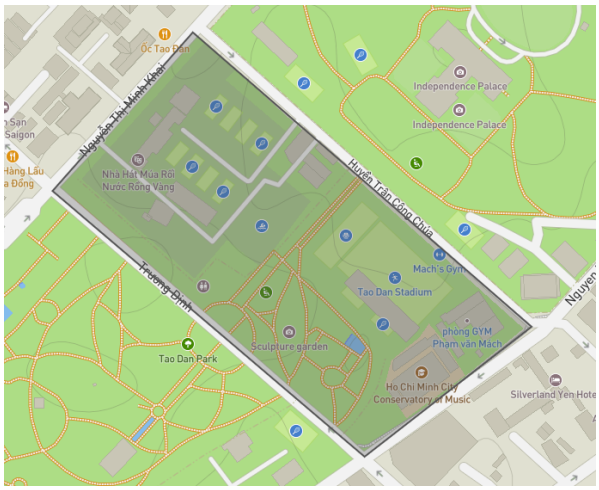
```
{
  "type": "Feature",
  "properties": {
    "PathName": "Ben Thanh-Thuan Loc"
  },
  "geometry": {
    "type": "LineString",
    "coordinates": [
      [106.7059326, 10.7695398],
      [106.7060699, 10.7695999],
      [106.7061615, 10.7697296],
      [106.7063903, 10.7705497],
      [106.7063598, 10.7707195],
      [106.7063369, 10.7709903],
      [106.7063903, 10.7716178],
      [106.7064437, 10.7722406],
      [106.7064971, 10.7728643],
      [106.7065506, 10.7734870],
      [106.7066040, 10.7741098],
      [106.7066574, 10.7740936]
    ]
  }
}
```



Listing 2.3: GeoJSON LineString

ⓓ | Polygon

```
{
  "type": "Feature",
    "properties": {
        "Name": "Tao Dan Park"
    },
    "geometry": {
      "coordinates": [
        [
          [106.6929551, 10.7773571],
          [106.6917032, 10.7759683],
          [106.6944880, 10.7736426],
          [106.6959784, 10.7747803],
          [106.6929551, 10.7773571]
        ]
      ],
      "type": "Polygon"
    }
  }
}
```



Listing 2.4: GeoJSON Polygon

```
{
  "type": "Feature",
  "geometry": {
    "coordinates": [
      [
        [106.6974673, 10.7440676],
        [106.6964724, 10.7441327],
        [106.6964787, 10.7436847],
        [106.6973864, 10.7435340],
        [106.6974673, 10.7440676]
      ], [
        [106.6966164, 10.7439634],
        [106.6966384, 10.7437807],
        [106.6969000, 10.7438044],
        [106.6970271, 10.7437115],
        [106.6972078, 10.7436630],
        [106.6972887, 10.7437105],
        [106.6973139, 10.7438798],
        [106.6972246, 10.7439510],
        [106.6966164, 10.7439634]
      ]
    ],
    "type": "Polygon"
  }
}
```

Listing 2.5: GeoJSON Polygon with holes. The resulting shape is obtained by uniting polygons with positive signed area ($A > 0$), then subtracting polygons with negative signed area ($A < 0$)

Chapter 3

Shapely: Spatial Analysis

This chapter is based mostly on the official Shapely documentation, Release 2.0.3 [7]

Shapely is a Python package for **set-theoretic analysis** and **manipulation of planar features**, wrapping the widely deployed GEOS library. As usual, installation can be done via `pip` or `conda`:

```
pip install shapely
```

Listing 3.1: Shapely: Installing Python `shapely` package

Shapely, by default, only uses the Cartesian coordinate system. For spatial analysis, it is practical to convert spherical coordinates to Cartesian coordinates, apply the calculations/transformations, then revert back to geographic coordinates. See Example: Converting from WGS-84 to VN-2000.

3.1 | Spatial Data Model

Fundamental geometric objects are characterised by three set of points, *interior*, *exterior* and *boundary* the union of which coincides with the plane.

ⓐ | Points: `Point` class

Definition of a Point:

- Interior: One point
- Boundary: $\emptyset$
- Exterior: All other points
- Dimension: 0

A Point is implemented in the `Point` class. Its 'collectional' version is the `MultiPoint` class.

```
class Point(*args):
    # A geometry type that represents a single coordinate
    # with x,y and possibly z values.
```

Listing 3.1: Shapely: `Point` class

```
>>> from shapely.geometry import *
>>> p = Point(3, 5)
>>> p
<POINT (3 5)>
>>> p.x, p.y
(3.0, 5.0)
>>> p.area, p.bounds
(0.0, (3.0, 5.0, 3.0, 5.0))
```

Listing 3.2: Shapely: Point

ⓑ | Curves: `LineString` class, `LinearRing` class

Definition of a Curve:

- Interior: All points along the curve's length
- Boundary: Two endpoints
- Exterior: All other points
- Dimension: 1

A Curve is implemented in the `LineString` class and the `LinearRing` class. Note that a smooth curve can only be approximated in this model.
Their collectional versions are `MultiLineString` and `MultiLinearRing`.

```
class LineString(coordinates=None):
    # A geometry type composed of one or more line segments.
```

Listing 3.3: Shapely: `LineString` class

```
>>> from shapely.geometry import *
>>> line = LineString([
...     [0, 0],
...     [0, 1],
...     [1, 1],
...     [1, 2],
...     [2, 2],
... ])
>>> line
<LINESTRING (0 0, 0 1, 1 1, 1 2, 2 2)>
>>> line.length
4.0
>>> line.area, line.bounds
(0.0, (0.0, 0.0, 2.0, 2.0))
```

Listing 3.4: Shapely: LineString

© | Surface: `Polygon` class

Definition of a Surface:

- Interior: All points within the curves
- Boundary: One or more curves
- Exterior: All other points, including those within **holes**
- Dimension: 2

A Surface is implemented in the `Polygon` class. Note that a smooth surface can only be approximated in this model.

```
class Polygon(shell=None, holes=None):
    # A geometry type representing an area that is enclosed by a
    # linear ring. It may have one or more negative-space holes
    # which are also bounded by linear rings
```

Listing 3.5: Shapely: `Polygon` class

```
>>> from shapely.geometry import *
>>> shell = [[0, 0], [0, 4], [4, 4], [4, 0]]
>>>  hole = [[1, 1], [1, 3], [3, 3], [3, 1]]
>>> poly = Polygon(shell=shell, hole=hole)
>>> poly
<POLYGON ((0 0, 0 4, 4 4, 4 0, 0 0), (1 1, 1 3, 3 3, 3 1, 1 1))>
>>> poly.area, poly.bounds
4.0
>>> poly.area, poly.bounds
(12.0, (0.0, 0.0, 4.0, 4.0))
```

Listing 3.6: Shapely: Polygon

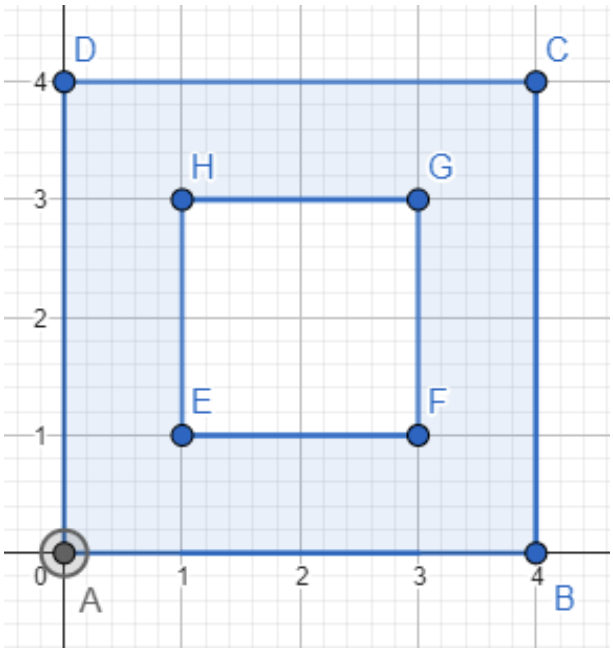


Figure 3.1: Shapely: The `Polygon` in the sample code is obtained by subtracting the polygon $ABCD$ by the polygon $EFGH$.

3.2 | Geometric Relationships

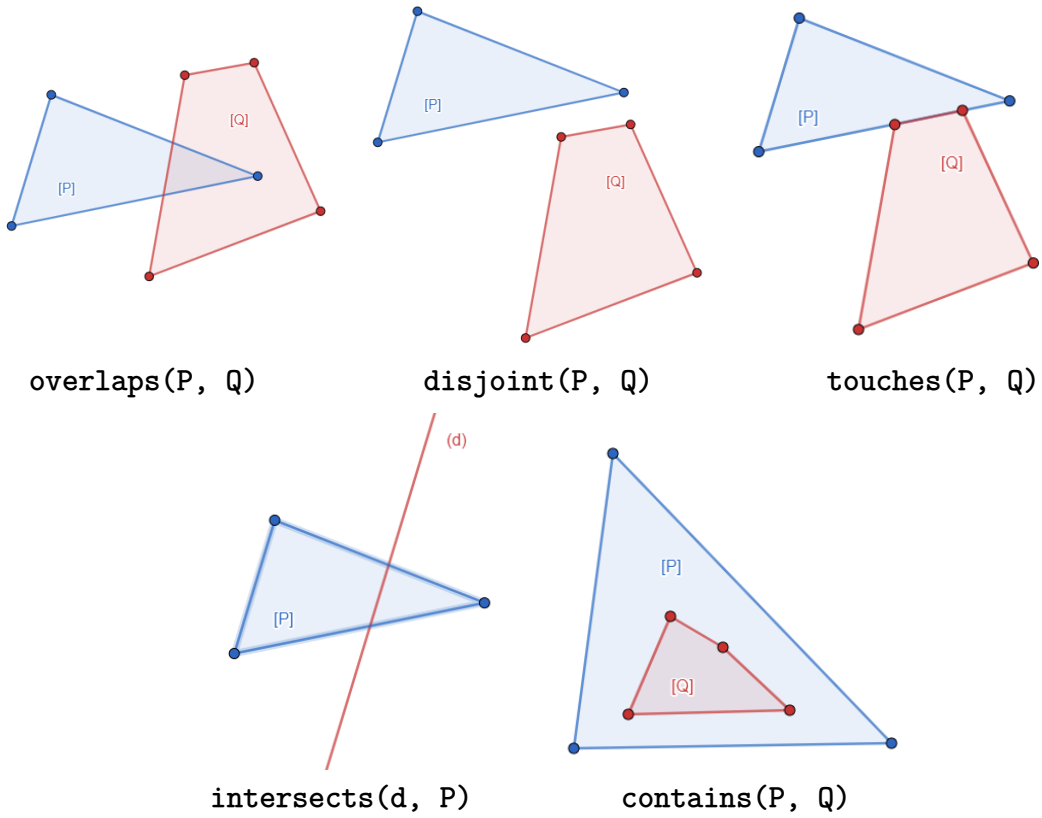
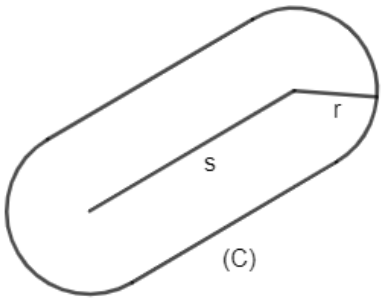
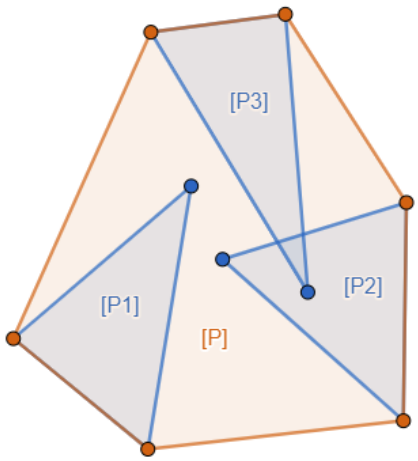


Figure 3.2: Shapely: Geometric relationships and their Shapely equivalent code snippet

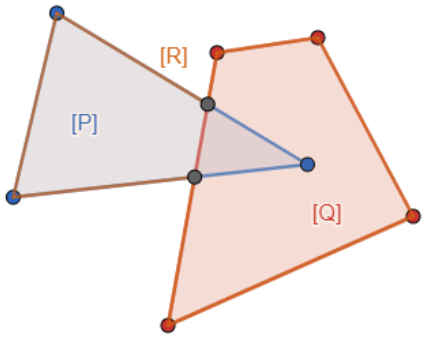
3.3 | Geometric Operations

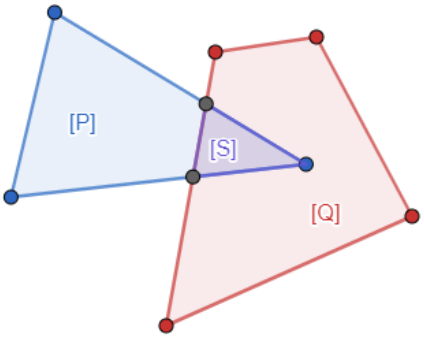





```
C = buffer(s, r)
```

```
P = convex_hull(MultiPolygon([P1, P2, P3]))
```






```
R = union(P, Q)
```

```
R = intersection(P, Q)
```

Figure 3.3: Shapely: Geometry operations and their Shapely equivalent code snippet

Chapter 4

R-tree: Data Structure for Spatial Data

This chapter is based mostly on the official R-tree Documentation, Release 1.2.0[8]

R-trees are tree data structures used in spatial analysis. Its most notable applications are finding the nearest point(s) and retrieving points within a given distance.

The R-tree data structure is based on the B-tree: Each leaf node is an object, nearby nodes are grouped and represented by their minimum bounding rectangle at a higher level node. Although having linear worst-case complexity for most operations, R-trees have been shown to perform well on real world data.

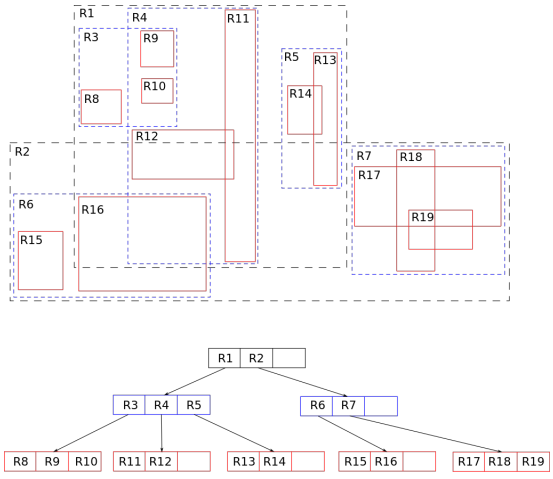


Figure 4.1: R-tree: Image of an R-tree. *(Source: Wikipedia)*

In this chapter we shall not dive deep into the theorical aspect of R-trees, rather we will introduce their applications via the `Rtree` Python package, a Python wrapper for libspatialindex, providing a number of advanced spatial indexing features, such as *intersection search* and *nearest neighbour search*. Installation can be done via `pip` or `conda`:

```
pip install rtree
```

Listing 4.1: R-tree: Installing Python `rtree` package

```
class rtree.index.Index:
    def __init__(self, *arg, **kwargs):
        """
        Creates a new R-tree Index object.
        """

    @property
    def bounds(coordinate_interleaved: bool) -> Iterable[int]:
        """
        Returns the bounds of the R-tree.
        """

    def insert(id: int, coodinates: Any, obj: object = None) -> None:
        """
        Insert a new object with id and coordinates.
        """

    def delete(id: int, coordinates: Any) -> None:
        """
        Delete an object with id and coordinates.
        """

    def count(coordinates: Any) -> int:
        """
        Returns the count of objects intersecting the given coordinates.
        """

    def intersection(coordinates: Any, object = Literal[X]) -> Iterator[Y]:
        """
        Returns ids/objects intersecting the given coordinates.
        """

    def nearest(coordinates: Any, num_results: int, object = Literal[X])
        -> Iterator[Y]:
        """
        Returns ids/objects of the k-nearest objects (k = num_results)
        """

    def close() -> None:
        """
        Frees R-tree from memory.
        """
```

Listing 4.2: R-tree: Implementation of `rTree.index.Index` class

4.1 | Creation

(a) | rtree.index.Index constructor

- `filename` / `stream`: Defines the file-based storage for the RTree. If left blank, RTree will be stored on default memory.
- `interleaved=True`
  Returns in the form `[Xmin, Ymin, ..., Xmax, Ymax, ...]` if `interleaved=True`, otherwise returns in the form `[Xmin, Xmax, Ymin, Ymax,...]`.
- `properties`: An `index.Property` object, containing creation and instantiation properties for the object.

For now, we will be using the blank constructor.

```
>>> from rtree import index
>>> index.Index()
rtree.index.Index(
    bounds=[
         1.7976931348623157e+308,  1.7976931348623157e+308,
        -1.7976931348623157e+308, -1.7976931348623157e+308
    ],
    size=0
)

>>> index.Index(interleaved=False)
rtree.index.Index(
    bounds=[
         1.7976931348623157e+308, -1.7976931348623157e+308,
         1.7976931348623157e+308, -1.7976931348623157e+308
    ],
    size=0
)
```

Listing 4.1: R-tree: Sample Code on Constructor

Note that in the above example, range_x = range_y = $(+\infty, -\infty) = \emptyset$.

4.2 | Insertion and Deletion

(a) | Insertion: Index.insert(id, coordinates, object)

Insert one item into the R-tree with parameters:

- `id`: The id of the object, and need not be unique.
- `coordinates`: A tuple of numbers defining the bounding rectangle.
- `obj`: An object associated with the coordinates.

The method does not guarantee uniqueness, therefore two objects with identical `id` and `coordinates` can co-exist. Uniqueness must be enforced by the user.

ⓑ | Deletion: `Index.delete(id, coordinates)`

Deletes ONE item from the R-tree with parameters:

- `id`: The id of the object, and need not be unique.
- `coordinates`: A tuple of numbers defining the bounding rectangle.

If multiple objects satisfy the criteria, only one item will be deleted.

```
>>> idx = index.Index()
>>> idx.insert(id=1, coordinates=(0, 0, 2, 2), obj="A")
>>> idx.insert(id=2, coordinates=(1, 1, 3, 3), obj="B")
>>> idx.insert(id=3, coordinates=(2, 2, 4, 4), obj="C")
>>> idx
rtree.index.Index(bounds=[0.0, 0.0, 4.0, 4.0], size=3)
>>> idx.delete(id=1, coordinates=(0, 0, 2, 2))
>>> idx          # Deletion succeeds, size decreases to 2
rtree.index.Index(bounds=[1.0, 1.0, 4.0, 4.0], size=2)
>>> idx.delete(id=2, coordinates=(0, 0, 2, 2))
>>> idx          # Deletion fails, size remains at 2
rtree.index.Index(bounds=[1.0, 1.0, 4.0, 4.0], size=2)
```

Listing 4.1: R-tree: Sample Code on Insertion and Deletion

4.3 | Querying

ⓐ | *property* `Index.bounds(interleaved=True)`

Returns the bounds of the RTree.

- **`coordinate_interleaved=True`**.

```
>>> idx
rtree.index.Index(bounds=[1.0, 1.0, 4.0, 4.0], size=2)
>>> idx.bounds
[1.0, 1.0, 4.0, 4.0]
```

Listing 4.1: R-tree: Sample Code on property `bounds`

ⓑ | `Index.count(coordinates)` `Index.intersection(coordinates, objects)`

Counts/Returns objects that intersects the given coordinates.

- **`coordinates`**. Coordinates in query
- **`objects = True | False | 'raw'`**. Describe the return type for the satisfied objects.

- If `True`, returns object wrapped in `rtree.index.Item` class
- If `False`, returns objects' `id`.
- If `'raw'`, returns objects in raw form (i.e., without the `Item` wrapper).

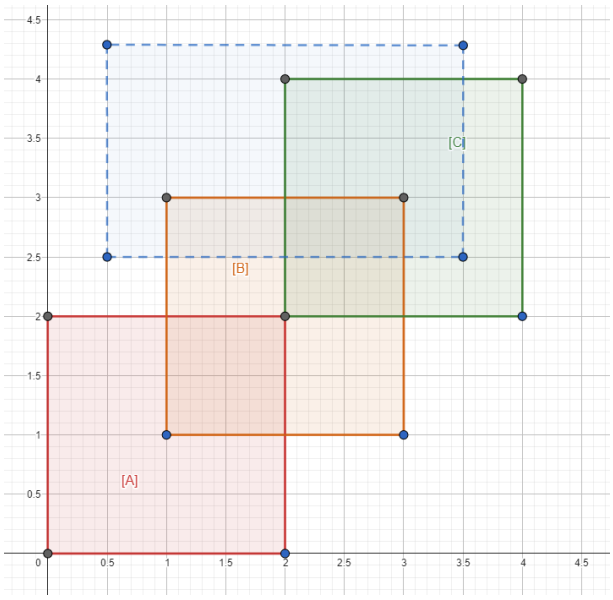


Figure 4.2: R-tree: Sample Code on Intersection operations *(Illustration)*

```
>>> idx = index.Index()
>>> idx.insert(id=1, coordinates=(0, 0, 2, 2), obj="A")
>>> idx.insert(id=2, coordinates=(1, 1, 3, 3), obj="B")
>>> idx.insert(id=3, coordinates=(2, 2, 4, 4), obj="C")
>>> query = (0.5, 2.5, 3.5, 4.5)
>>> idx.count(query)
2
>>> idx.intersection(query)
<generator object Index._get_ids at 0x0000026337738590>
>>> list(idx.intersection(query))
[2, 3]
>>> list(idx.intersection(query, objects='raw'))
['B', 'C']
>>> list(idx.intersection(query, objects=True))
[
    <rtree.index.Item object at @address1>,
    <rtree.index.Item object at @address2>
]
>>> [item.object for item in idx.intersection(query, objects=True)]
['B', 'C']
```

Listing 4.2: R-tree: Sample Code on Intersection operations

ⓒ | Index.nearest(coordinates, num_objects, objects)

Runs the k-nearest neighbour algorithm.

- **coordinates**. Coordinates in query
- **num_results=1**. The number of objects to return nearest to the coordinates. It is the parameter k in the $k-$nearest neighbour algorithm.
- **objects = True | False | 'raw'**. Describe the return type for the satisfied objects, similar to what in subsection ⓑ.

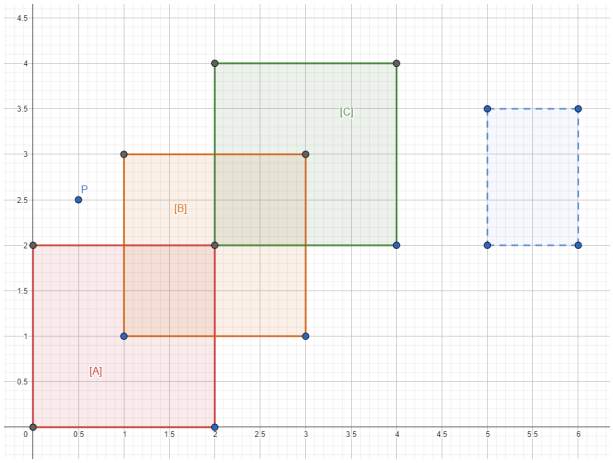


Figure 4.3: R-tree: Sample Code on Nearest Neighbour *(Illustration)*

```
>>> idx = index.Index()
>>> idx.insert(id=1, coordinates=(0, 0, 2, 2), obj="A")
>>> idx.insert(id=2, coordinates=(1, 1, 3, 3), obj="B")
>>> idx.insert(id=3, coordinates=(2, 2, 4, 4), obj="C")
>>> list(idx.nearest((5, 1)))
[3]
>>> list(idx.nearest((0.5, 2.5), objects='raw'))
['B', 'A']
>>> list(idx.nearest((5, 2, 6, 3.5), num_results=2, objects='raw'))
['C', 'B']
```

Listing 4.3: R-tree: Sample Code on Nearest Neighbour

ⓓ | Index.close()

Disposes the RTree, frees up memory.

Chapter 5

LangChain: Apply Large Language Model to Querying Databases

This chapter is based mostly on the official Python Langchain Documentation[9]

LangChain is a framework for developing applications powered by language models. LangChain provides the ability for the program to be:

- **Context-aware**: connect a language model to sources of context
- **Reasonable**: rely on a language model to reason (answering queries, deciding actions, etc.)

This report will briefly discuss the application of LangChain only to the task of querying databases. *It is to be noticed that Large Language Models (LLMs) are still at their early stages, thus most LLM libraries expect users to use with special care and revision. That being said, LLMs do bring a lot of potential to empower existing programs and enable them to do tasks that have never been thought of before.*

To install LangChain, install the following dependencies:

```
pip install langchain
pip install langchain_experimental
pip install langchain_openai
```

Querying a `pandas.DataFrame`

LangChain supports querying a `pandas.DataFrame` directly through the `langchain_experimental.agents.agent_toolkits` module.

```
>>> import pandas as pd

>>> from langchain.agents.agent_types import AgentType
>>> from langchain_experimental.agents.agent_toolkits \
...     import create_pandas_dataframe_agent
>>> from langchain_openai import OpenAI
```

Listing 5.1: LangChain: Importing libraries

Let say we have a `DataFrame` of all *Bus stops* in Ho Chi Minh City called `stops_df`.

```
>>> stops_df = pd.read_csv('stops.csv')
>>> stops_df.sample(5)
      StopId     Code          Zone
388      483    Q9 213       Quan 9
5852     478    Q2 072       Quan 2
8575    1128   HNB 048  Huyen Nha Be
8225    3559  QCCT128  Huyen Cu Chi
4254     841    Q7 102       Quan 7
```

Listing 5.2: LangChain: Importing libraries

Now we create an ”agent” that will link our dataframe with a LLM.

```
>>> agent = create_pandas_dataframe_agent(
...     OpenAI(temperature=0, openai_api_key=OPENAI_API_KEY),
...     stops_df,
...     verbose=True
... )
```

Listing 5.3: LangChain: Create an ”agent”

Note that we can replace the `OpenAI` model by any large language models, most of which are included in the `langchain.llms` module.

Asking a question is as simple as ”invoking” the agent.

```
>>> agent.invoke("List 5 stops in District 5")
```

Listing 5.4: LangChain: ”Invoke” a query to get the answer

```
> Entering new AgentExecutor chain...
Thought: I need to filter the dataframe for stops in District 5
Action: python_repl_ast
Action Input: df[df['Zone'] == 'Quan 5'].head()    StopId    Code    Zone
93      569  Q5 036  Quan 5
94      573  Q5 035  Quan 5
95      433  Q5 017  Quan 5
96      434  Q5 018  Quan 5
97      436  Q5 019  Quan 5I now know the final answer
Final Answer: The 5 stops in District 5 are:
1. StopId: 569, Code: Q5 036
2. StopId: 573, Code: Q5 035
3. StopId: 433, Code: Q5 017
4. StopId: 434, Code: Q5 018
5. StopId: 436, Code: Q5 019
```

Listing 5.5: LangChain: Response from the agent

And the method will return a JSON object of the user input and the model final output of the query.

```
{'input': 'List 5 stops in District 5',
'output': 'The 5 stops in District 5 are:\n1. StopId: 569, \
Code: Q5 036\n2. StopId: 573, Code: Q5 035\n3. StopId: \
433, Code: Q5 017\n4. StopId: 434, Code: Q5 018\n5. \
StopId: 436, Code: Q5 019'}
```

Listing 5.6: LangChain: Final input and output

Chapter 6

Elements of a Bus Network

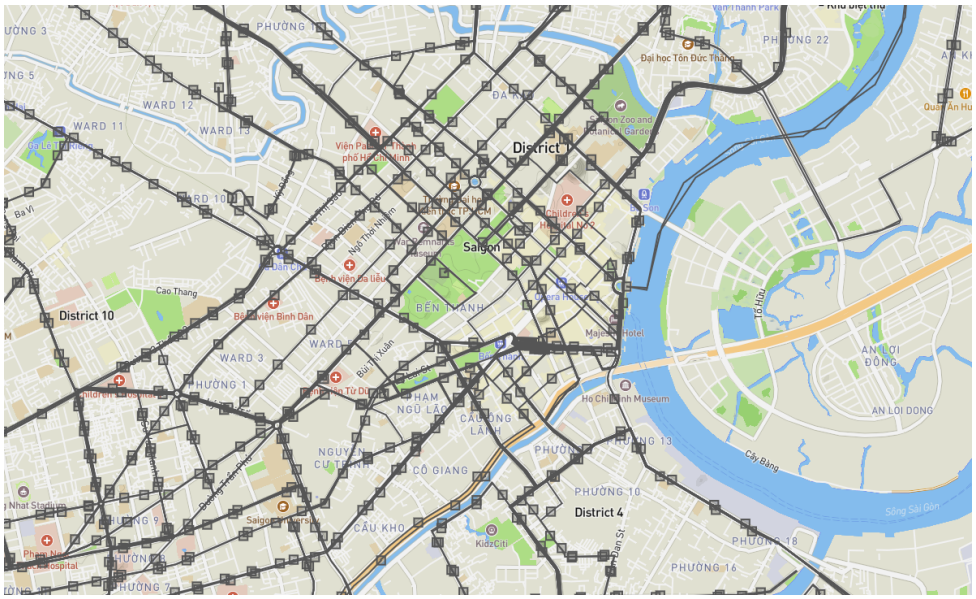


Figure 6.1: Elements of a Bus Network: The public bus network in Ho Chi Minh City. Squares represent bus stops, paths represent bus routes

6.1 | Stop (the stop module)

*Stop*s are the basic elements of a bus network. It defines places where buses stop temporarily to let passengers get on or off the bus.

ⓐ | Properties

- ***property*** `stop_id: int`
- ***property*** `code: str`
- ***property*** `name: str`
- ***property*** `stop_type: str`
- ***property*** `zone: str`
- ***property*** `ward: str`
- ***property*** `address_no: str`
- ***property*** `street: str`
- ***property*** `support_disability: bool`
- ***property*** `status: str`
- ***property*** `latitude: float`
  property `longtitude: float`
  Coordinates of the bus stop in WGS-84 coordinate system.
- ***property*** `coord`
  Coordinates of the bus stop in VN-2000 coordinate system.
- ***property*** `search: list[str]`
  List of tokens that can be used to search for the bus stop
- ***property*** `routes: list[str]`
  List of route names crossing the bus stop

ⓑ | Methods

- `to_string() -> str`
  Display information of a `Stop` in a string format.
  This function is called by `__repr__()`.
- ***static*** `from_dict(obj: dict) -> Stop`
  Create `Stop` from a Python dictionary.
- `to_dict() -> dict`
  Convert `Stop` to a Python dictionary.
- ***static*** `from_json(file: str) -> Stop`
  Create `Stop` from JSON file. The function loads a JSON file into a Python dictionary, then call `from_dict()`.

- to_json(file: str) -> None
  Export Stop information to JSON file. The function calls to_dict() to convert Stop to a dictionary, then dumps it into a JSON file.

```
>>> from stop import Stop
>>> stop = Stop.from_json('stop.json')
>>> stop.to_string()
================[Nguyen Van Linh]=================
| StopID:              7182                       |
| Code:                Q7 BD2                     |
| Name:                Nguyen Van Linh            |
| Type:                O son                      |
| Zone:                Quan 7                     |
| Ward:                Phuong Tan Phong           |
| Address No.:         R1-49                      |
| Street:              Bui Bang Doan              |
| Support Disability:  True                       |
| Status:              Dang khai thac             |
| Lng:                 106.708499                 |
| Lat:                 10.729471                  |
| Search tokens:       NVL, R, BBD                |
| Routes:              D2                         |
==================================================
>>> stop.routes.append('D3')
>>> stop.to_json('out.json')
>>> stop2 = Stop.from_json('out.json')
>>> stop2
================[Nguyen Van Linh]=================
| StopID:              7182                       |
| Code:                Q7 BD2                     |
| Name:                Nguyen Van Linh            |
| Type:                O son                      |
| Zone:                Quan 7                     |
| Ward:                Phuong Tan Phong           |
| Address No.:         R1-49                      |
| Street:              Bui Bang Doan              |
| Support Disability:  True                       |
| Status:              Dang khai thac             |
| Lng:                 106.708499                 |
| Lat:                 10.729471                  |
| Search tokens:       NVL, R, BBD                |
| Routes:              D2 -> D3                   |
==================================================
```

Listing 6.1: Elements of a Bus Network: Example of a Stop object

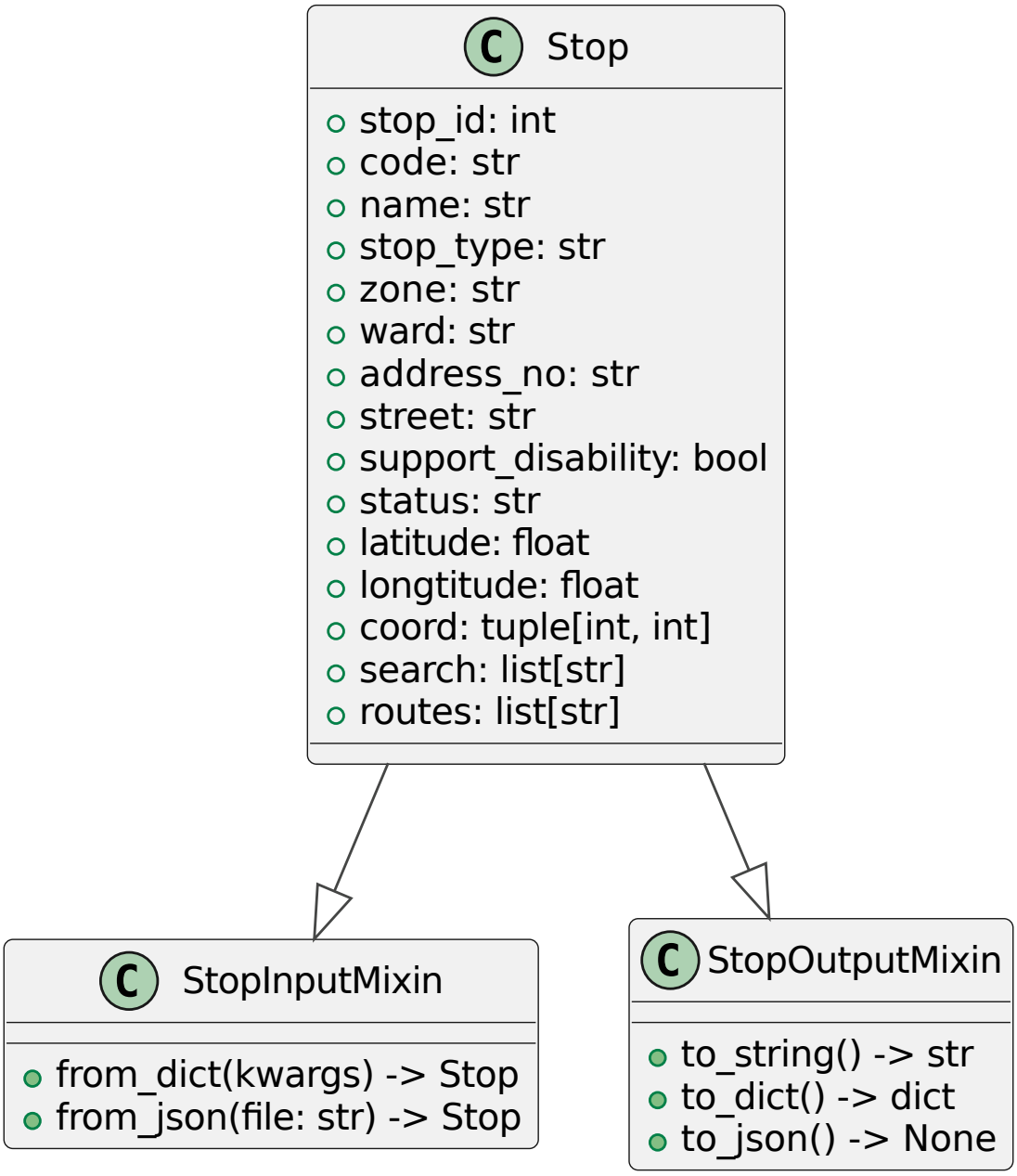


Figure 6.2: Elements of a Bus Network: Stop class diagram

6.2 | Variant (the variant module)

*Variant*s represent bus routes in a specific direction. A bus route has two variants, outbound and inbound. Some bus routes may only have one variant.

ⓐ | Properties

- ***property*** `route_id: int`
  property `route_var_id: int`
  property `route_ids: tuple[int, int]`
  IDs of a `Variant`.
  A route has two variants, each shares the same `route_id` but different `route_var_id`. `route_ids` is a tuple of `route_id` and `route_var_id`.
- ***property*** `number: str`
  Number of the bus variant. Must be of `str` type to store values such as `61-1, D2`.
- ***property*** `name: str`
  property `short_name: str`
  Name/Short name of the bus variant.
- ***property*** `start_stop: str`
  property `end_stop: str`
  Starting and ending stops of a variant
- ***property*** `distance: float`
  property `running_time: float`
  Distance and running time of the bus variant.

ⓑ | Methods

- `to_string() -> str`
  Display information of a `Variant` in a string format.
  This function is called by `__repr__()`.
- ***static*** `from_dict(obj: dict) -> Variant`
  Create `Variant` from a Python dictionary.
- `to_dict() -> dict`
  Convert `Variant` to a Python dictionary.
- ***static*** `from_json(file: str) -> Variant`
  Create `Variant` from JSON file. The function loads a JSON file into a Python dictionary, then call `from_dict()`.
- `to_json(file: str) -> None`
  Export `Variant` information to JSON file. The function calls `to_dict()` to convert `Variant` to a dictionary, then dumps it into a JSON file.

```
>>> from variant import Variant
>>> var = Variant.from_json('var.json')
>>> var
========[Route D2, Paris Baguette -> Cressent mall]========
| RouteNo:        D2                                       |
| StartStop:      Paris Baguette                           |
| EndStop:        Cressent mall                            |
| Name:           Luot di                                  |
| ShortName:      Luot di                                  |
| Distance:       3677.0000000000005                       |
| RunningTime:    14                                       |
| RouteIds:       (212, 1)                                 |
============================================================
>>> var.route_id
212
>>> var.route_var_id
1
>>> var.route_ids
(212, 1)
>>> var.to_dict()
{
    'RouteNo': 'D2',
    'StartStop': 'Paris Baguette',
    'EndStop': 'Cressent mall',
    'Name': 'Luot di'
    'ShortName': 'Luot di'
    'Distance': 3677.0000000000005,
    'RunningTime': 14,
    'RouteIds': (212, 1)
}
```

Listing 6.1: Elements of a Bus Network: Example of a `Variant` object

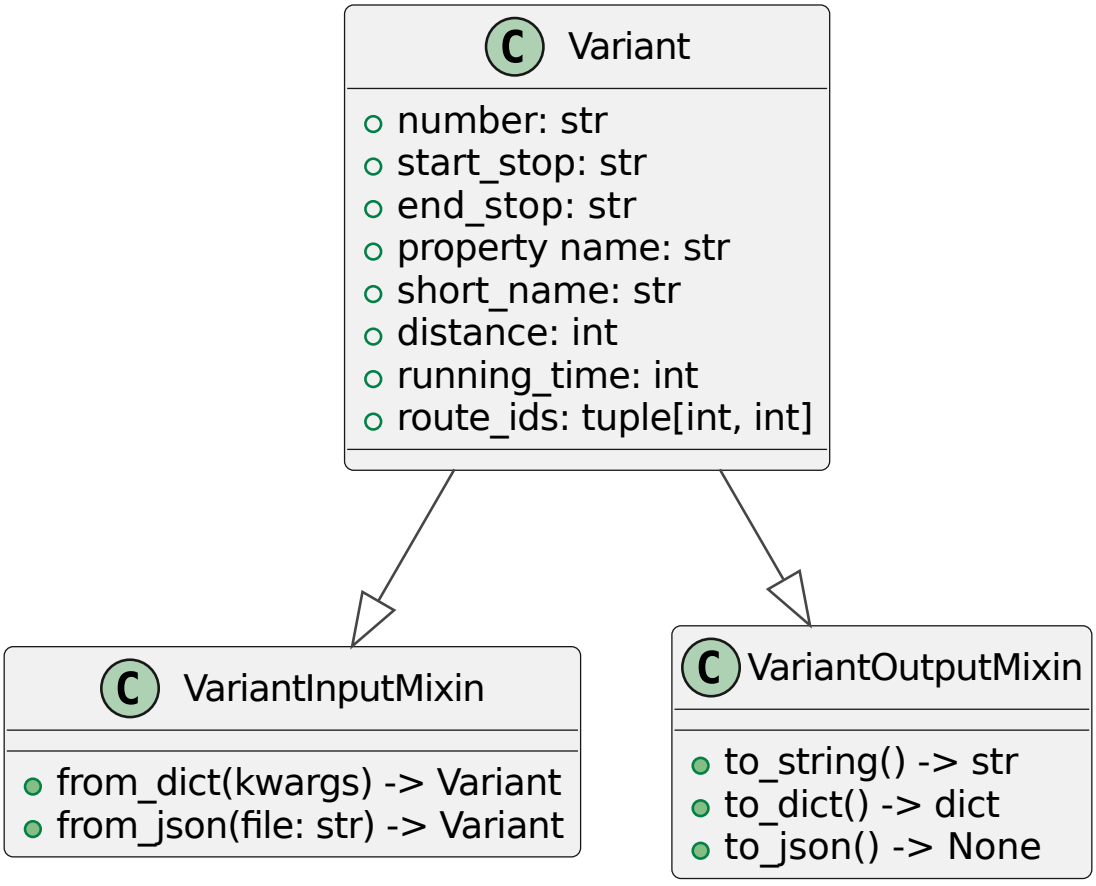


Figure 6.3: Elements of a Bus Network: Variant class diagram

6.3 | Path (the path module)

*Path*s represent the specific path of a bus variant in the form of a LineString.

ⓐ | Properties

- ***property*** `route_id: int`
  property `route_var_id: int`
  property `route_ids: tuple[int, int]`
  IDs of the route that the `Path` represents.
  A route has two variants, each shares the same `route_id` but different `route_var_id`. `route_ids` is a tuple of `route_id` and `route_var_id`.
- ***property*** `coords: list[tuple[float, float]]` Coordinates of a `LineString` representing the path of the bus variant.

ⓑ | Methods

- `polysides() -> Iterable[tuple[tuple[float, float], tuple[float, float]]]`
  Return an `Iterable` of tuples of coordinates, representing each segments of the `LineString`.
- `to_string() -> str`
  Display information of a `Path` in a string format.
  This function is called by `__repr__()`.
- ***static*** `from_dict(obj: dict) -> Path`
  Create `Path` from a Python dictionary.
- `to_dict() -> dict`
  Convert `Path` to a Python dictionary.
- ***static*** `from_json(file: str) -> Path`
  Create `Path` from JSON file. The function loads a JSON file into a Python dictionary, then call `from_dict()`.
- `to_json(file: str) -> None`
  Export `Path` information to JSON file. The function calls `to_dict()` to convert `Path` to a dictionary, then dumps it into a JSON file.

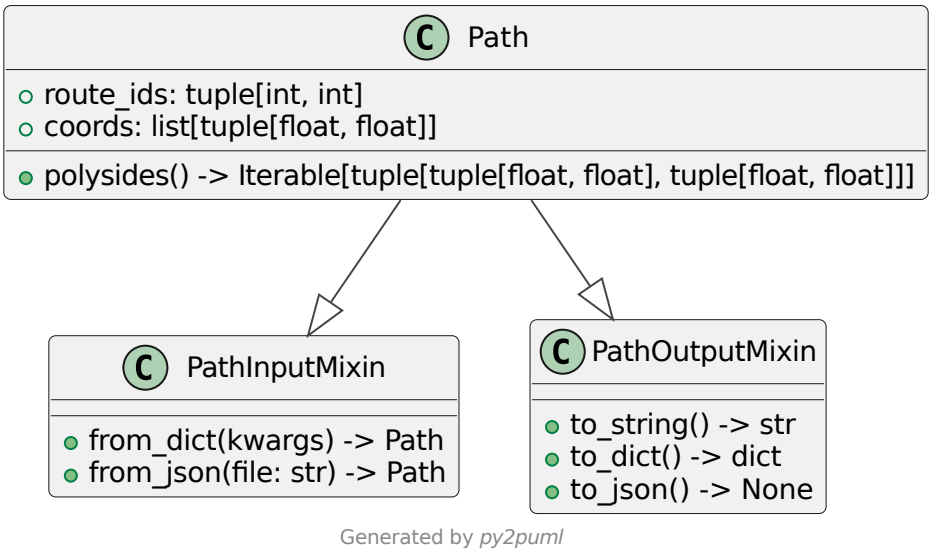


Figure 6.4: Elements of a Bus Network: Path class diagram

```
>>> from path import Path
>>> import math
>>> path = Path.from_json('path.json')
>>> path.route_ids
(212, 1)
>>> len(path.coords)
30
>>> sum(math.dist(p1, p2) for (p1, p2) in path.polysides())
3664.914478222333
```

Listing 6.1: Elements of a Bus Network: Example of a `Path` object

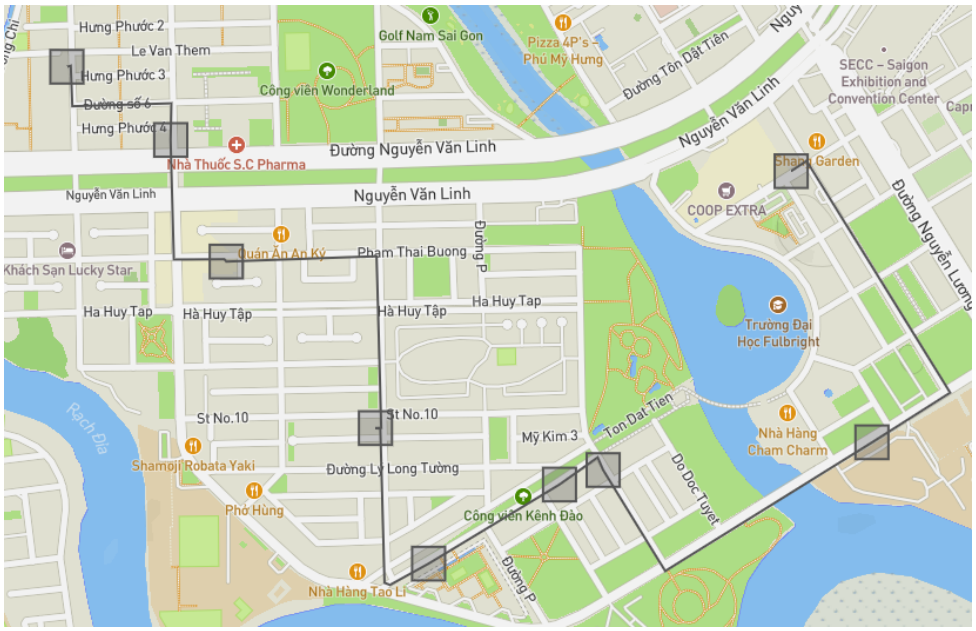


Figure 6.5: Elements of a Bus Network: The variant from *Paris Baguette* to *Cressent mall* has a total distance of approx. 3665km, whilst the distance in database is 3677km. Their relative difference is 0.33%.

Chapter 7

Querying List of Objects

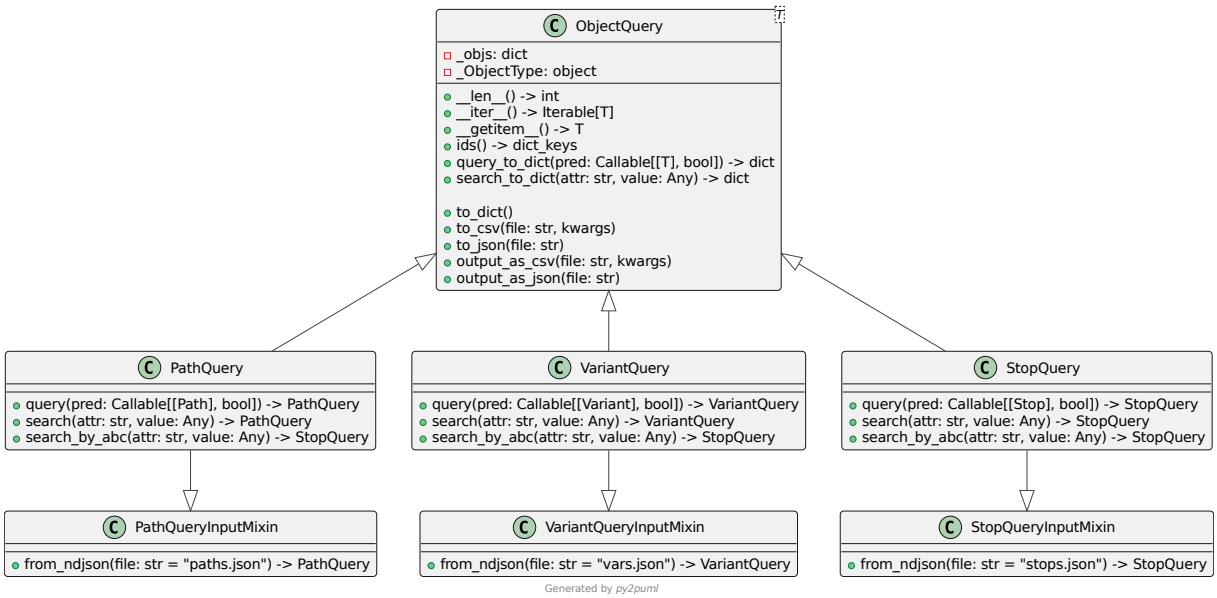


Figure 7.1: Querying List of Objects: ObjectQuery base class and derived classes for specific querying data types

7.1 | A generic object querying type: `ObjectQuery`

ⓐ | Properties

- `_objs: dict`
  Items in query are stored in a dictionary with their associated IDs.
- `_ObjectType: object`
  Type of items.
- ***property*** `ids: list[int]`
  Return ids of items.
- ***property*** `values: list[Any]`
  Return values of items.

(b) | Methods

- `query_to_dict(pred: Callable) -> dict`
  Query all items satisfying the predicate `pred`, return as a dictionary.
- `search_to_dict(attr: str, value: Any) -> dict`
  Search for all items with attributes matching `value`, return as a dictionary.

```
def search_to_dict(self, attr: str, value: Any) -> dict:
    if not hasattr(self._ObjectType, attr):
        raise AttributeError

    return self.query_to_dict(lambda obj: getattr(obj, attr) == value)
```

- `to_dict() -> dict`
  Returns internal state `_objs`.
- `to_json(file: str) -> None`
  Export `ObjectQuery` information to JSON file. The function dumps the dictionary `_objs` into a JSON file.
- `to_csv(file: str) -> None`
  Export `ObjectQuery` information to CSV file. The function converts the dictionary `_objs` into a `pandas` table before exporting to CSV.
- `output_as_json = to_json`
  `output_as_csv = to_csv`
  Aliases for `to_json` and `to_csv` functions.

7.2 | Inheritence from `ObjectQuery` for specific object types

(a) | StopQuery

```
class StopQuery(ObjectQuery, StopQueryInputMixin):
    def __init__(self, objs):
        super.__init__(objs=objs, ObjectType=Stop)

    def query(self, pred: Callable[[Stop], bool]):
        return StopQuery(self.query_to_dict(pred))

    def search(self, attr: str, value: Any):
        return StopQuery(self.search_to_dict(attr, value))

    # aliases
    search_by_abc = search
```

Listing 7.1: Querying List of Objects: `StopQuery` class

ⓑ | VariantQuery

```python
class VariantQuery(ObjectQuery, VariantQueryInputMixin):
    def __init__(self, objs):
        super.__init__(objs=objs, ObjectType=Variant)

    def query(self, pred: Callable[[Variant], bool]):
        return VariantQuery(self.query_to_dict(pred))

    def search(self, attr: str, value: Any):
        return VariantQuery(self.search_to_dict(attr, value))

    # aliases
    search_by_abc = search
```

Listing 7.2: Querying List of Objects: `VariantQuery` class

ⓒ | PathQuery

```python
class PathQuery(ObjectQuery, PathQueryInputMixin):
    def __init__(self, objs):
        super.__init__(objs=objs, ObjectType=Path)

    def query(self, pred: Callable[[Path], bool]):
        return PathQuery(self.query_to_dict(pred))

    def search(self, attr: str, value: Any):
        return PathQuery(self.search_to_dict(attr, value))

    # aliases
    search_by_abc = search
```

Listing 7.3: Querying List of Objects: `PathQuery` class

7.3 | Example use of ObjectQuery-derived class

```
>>> stops = StopQuery.from_ndjson()
>>> variants = VariantQuery.from_ndjson()
>>> paths = PathQuery.from_ndjson()
>>> print('There are {} stops, {} variants and {} paths.'\
...       .format(len(stops), len(variants), len(paths)))
There are 4397 stops, 297 variants and 297 paths.
>>> stop_id = 1234
>>> print(stops.search('stop_id', stop_id).values)
[==================[Nga 3 Cu Cai]==================
| StopID:              1234                       |
| Code:                HHM 053                    |
| Name:                Nga 3 Cu Cai               |
| Type:                Nha cho                    |
| Zone:                Huyen Hoc Mon              |
| Ward:                Xa Xuan Thoi Dong          |
| Address No.:         43/1 (ke 33/5A), 7/2A      |
| Street:              Quoc lo 22                 |
| Support Disability:  True                       |
| Status:              Dang khai thac             |
| Lng:                 106.603324                 |
| Lat:                 10.860602                  |
| Search tokens:       N3CC, 43/1(33/5A),7/2A, Ql22
| Routes:              122 -> 13 -> 62-5 -> 70-3 -> 74 -> 85 -> 94
==================================================]
>>> variants.query(lambda var: var.distance <= 1680)
===========[Route HS-73, Rung Sac -> tieu Hoc An Nghia]===========
| RouteNo:           HS-73                                      |
| StartStop:         Rung Sac                                   |
| EndStop:           tieu Hoc An Nghia                          |
| Name:              Luot di                                    |
| ShortName:         Truong Tieu Hoc An Nghia                   |
| Distance:          1680.0                                     |
| RunningTime:       20                                         |
| RouteIds:          (293, 1)                                   |
====================================================================
===========[Route HS-73, tieu Hoc An Nghia -> Rung Sac]===========
| RouteNo:           HS-73                                      |
| StartStop:         tieu Hoc An Nghia                          |
| EndStop:           Rung Sac                                   |
| Name:              Luot ve                                    |
| ShortName:         Rung Sac                                   |
| Distance:          1616.0                                     |
| RunningTime:       20                                         |
| RouteIds:          (293, 2)                                   |
====================================================================
```

Bibliography

[1] Source code for `pyproj.transformer`. [Online]. Available: https://pyproj4.github.io/pyproj/stable/api/transformer.html#pyproj-transformer

[2] PROJ contributors, "PROJ coordinate transformation software library," Open Source Geospatial Foundation, 2024. [Online]. Available: https://proj.org/

[3] `pyproj`. [Online]. Available: https://pypi.org/project/pyproj/

[4] "Decision No. 83/2000/QD-TTg on the use of Vietnam's National references and Coordinates Systems," July 2000. [Online]. Available: https://chinhphu.vn/default.aspx?pageid=27160&docid=7812

[5] [Online]. Available: https://geojson.org/

[6] H. Butler, M. Daly, A. Doyle, S. Gillies, T. Schaub, and S. Hagen, "The GeoJSON Format," RFC 7946, Aug. 2016. [Online]. Available: https://www.rfc-editor.org/info/rfc7946

[7] [Online]. Available: https://shapely.readthedocs.io/_/downloads/en/stable/pdf/

[8] [Online]. Available: https://readthedocs.org/projects/rtree/downloads/pdf/latest/

[9] "Langchain." [Online]. Available: https://python.langchain.com