Simple and Fast OGC API Servers for PostGIS with TiFeatures and TiMVT

Vincent Sarago - Developmentseed

Vincent sarago

Software Engineer @ Developmentseed

COG Tzar

Self-Taught Python dev with great mentors 👋 @sgillies

Creator of @RemotePixel

MSc in Earth Sciences

Bike & Coffee

TiTiler

https://developmentseed.org/titiler/

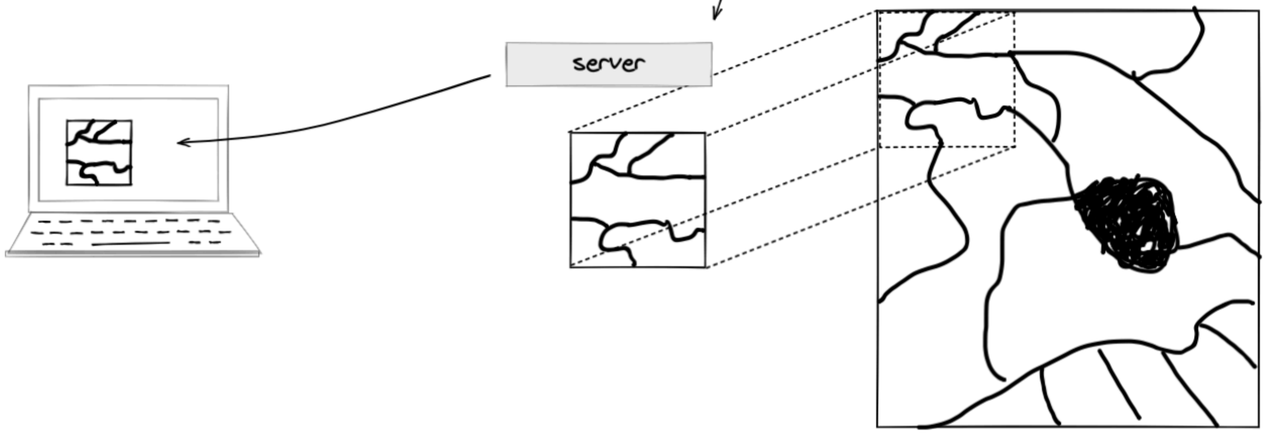
python>=3.8

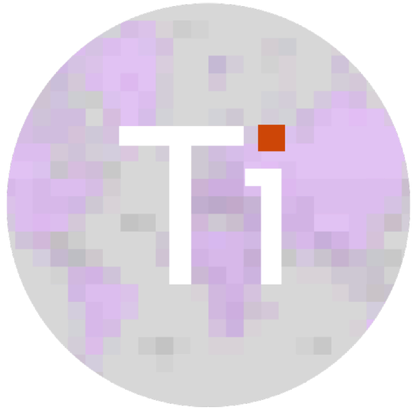
GDAL

FastAPI

Dynamic Raster Tile Services

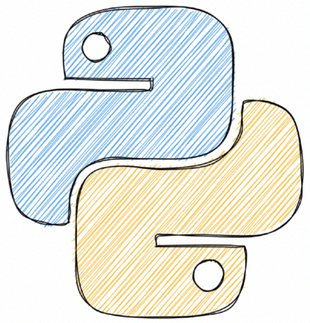
Processing:
- reprojection
- rescaling
- math
- ...





TiMVT

https://developmentseed.org/timvt



python>=3.8

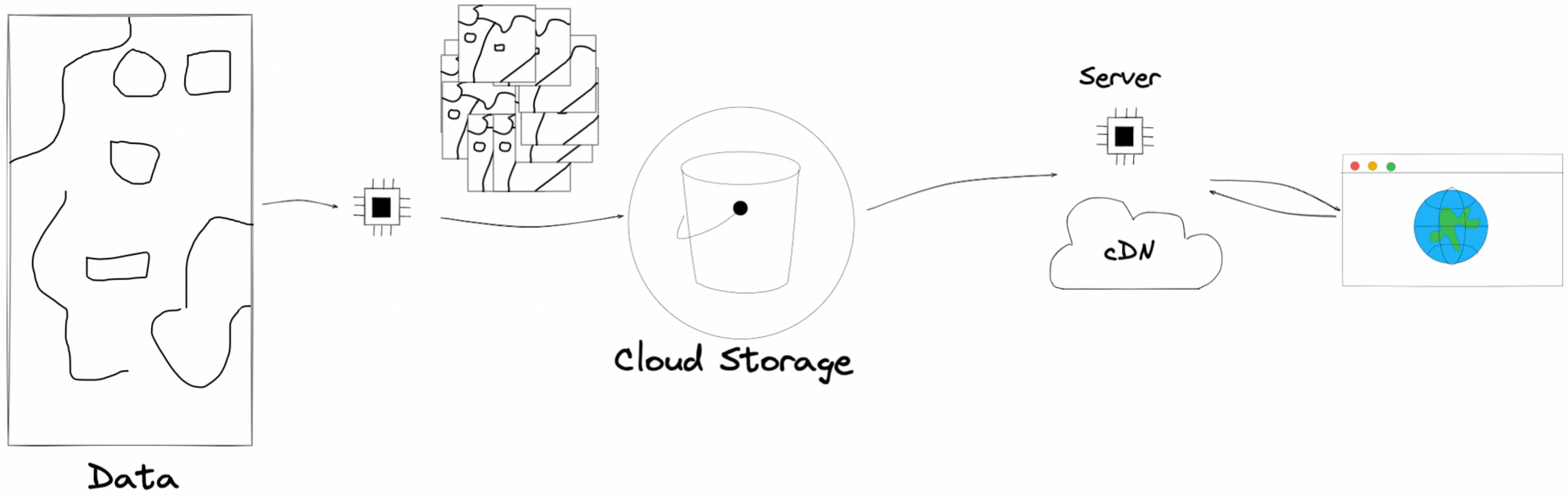
FastAPI

PyProj
Morecantile
Pydantic

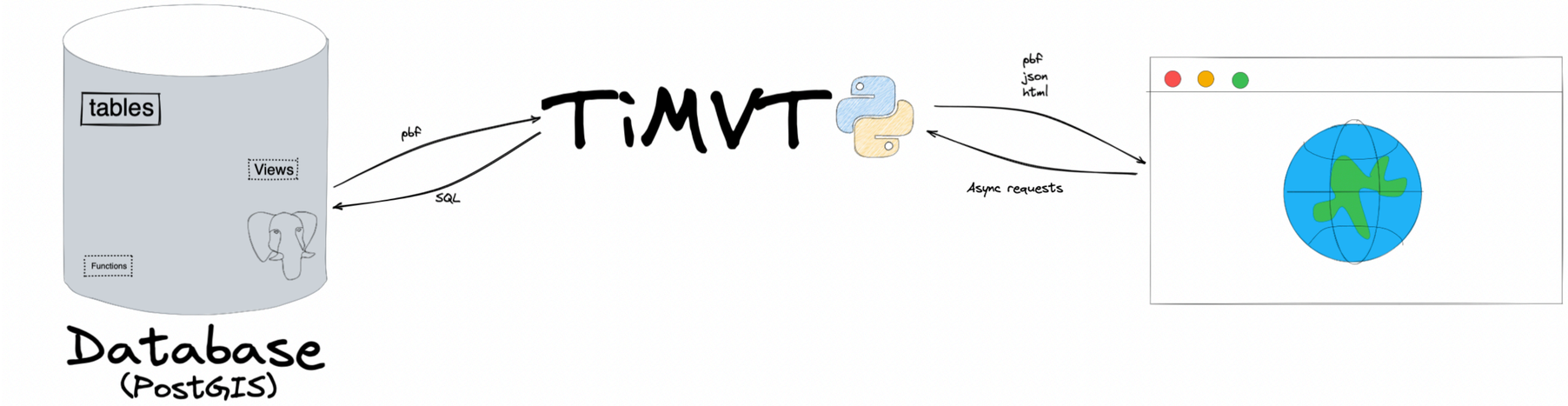
Map Tile (vector)

https://github.com/mapbox/vector-tile-spec

Static (pre-rendered)



Dynamic (rendering on-demand)

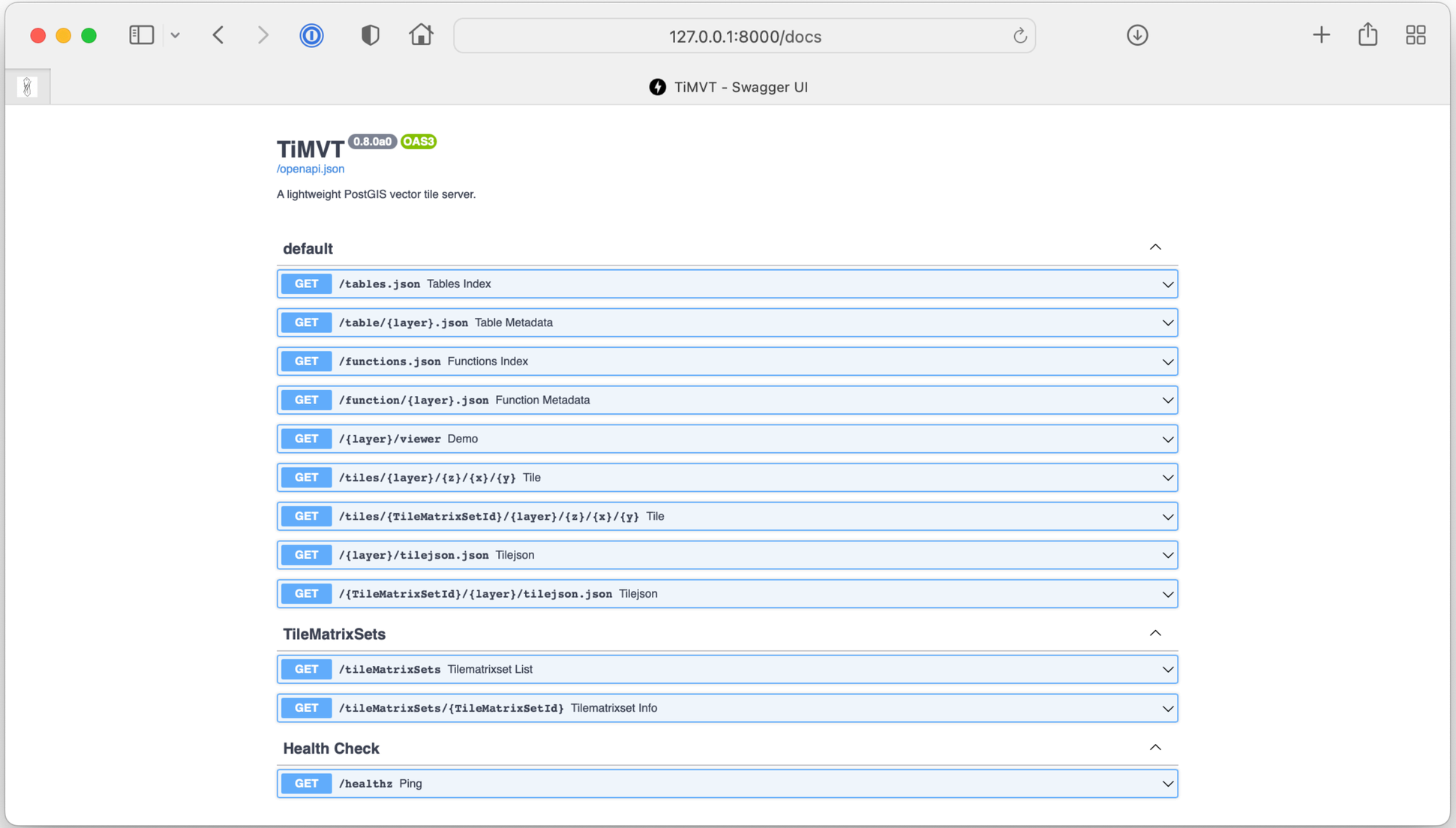


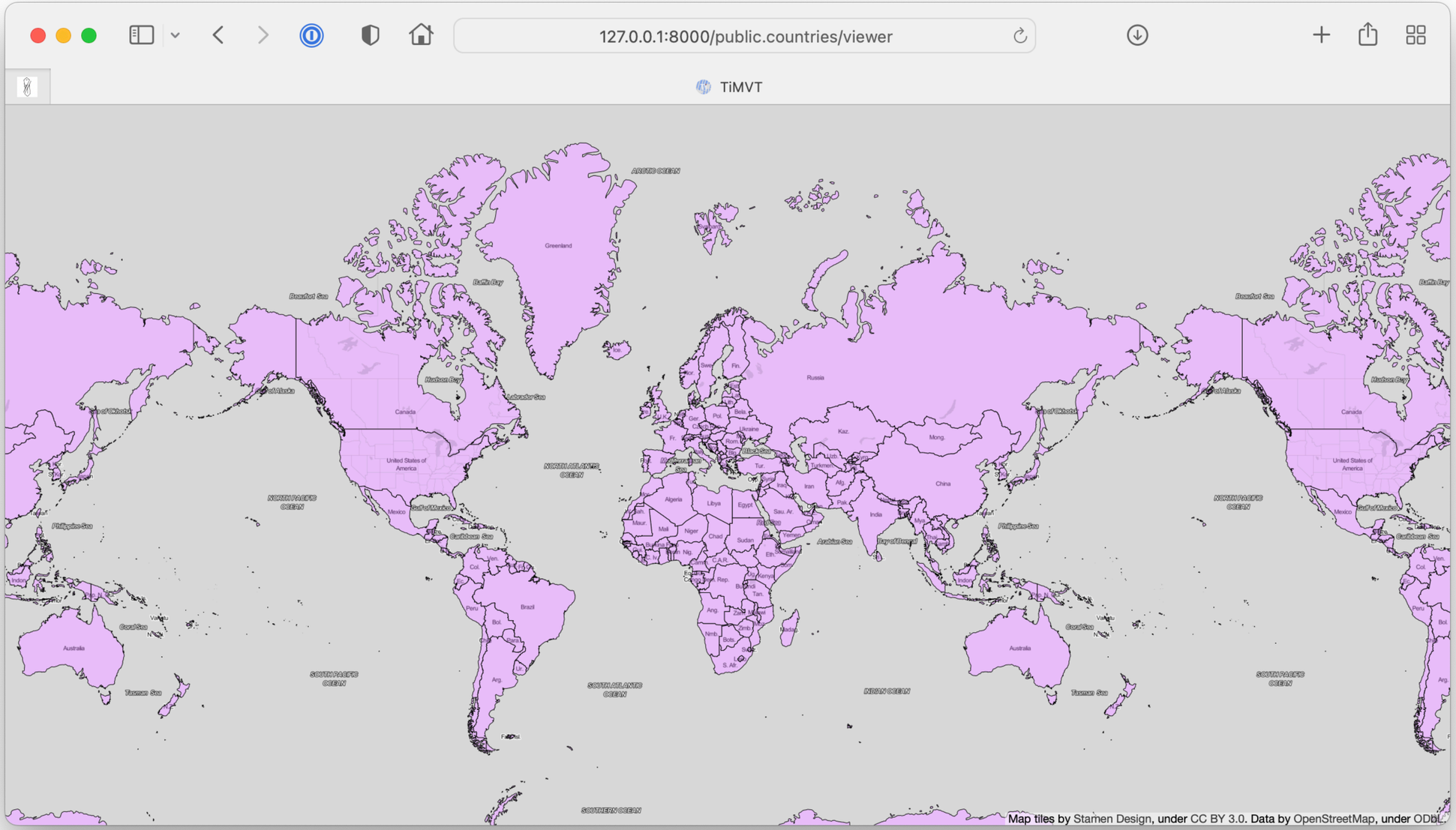
Features

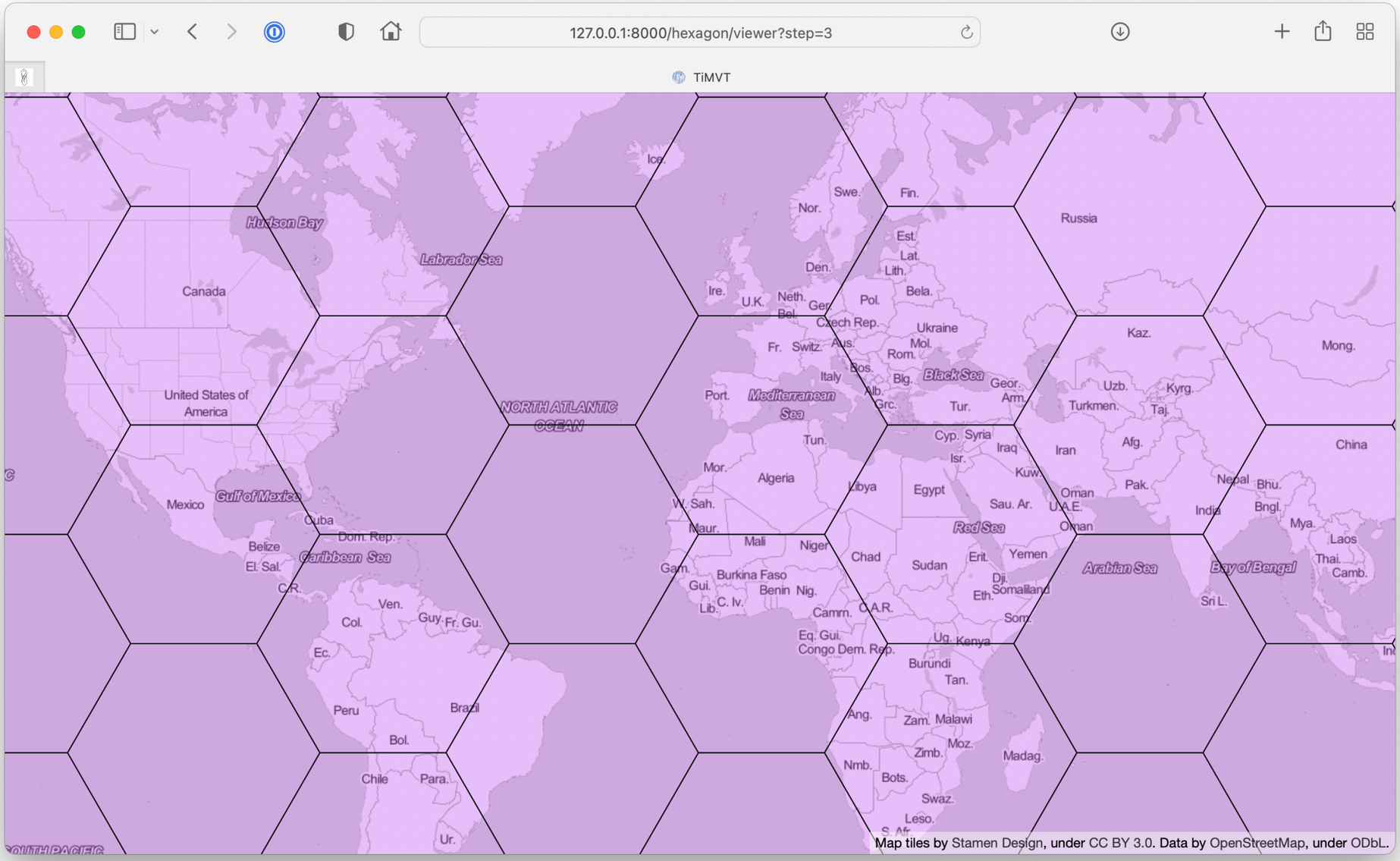
- Built with FastAPI
- Easy integration into user's application
- Async API (asyncpg)
- Table and Function layers
- Multiple TileMatrixSets via morecantile
- Web Viewer
- Only need `SELECT` privileges
- Functions are not written to the database

Don't fork it, Import it!

```
python -m pip install timvt
uvicorn timvt.main:app --port 8000
```


127.0.0.1:8000/docs
TiMVT - Swagger UI
TiMVT 0.8.0a0 OAS3
/openapi.json
A lightweight PostGIS vector tile server.
default
GET /tables.json Tables Index
GET /table/{layer}.json Table Metadata
GET /functions.json Functions Index
GET /function/{layer}.json Function Metadata
GET /{layer}/viewer Demo
GET /tiles/{layer}/{z}/{x}/{y} Tile
GET /tiles/{TileMatrixSetId}/{layer}/{z}/{x}/{y} Tile
GET /{layer}/tilejson.json Tilejson
GET /{TileMatrixSetId}/{layer}/tilejson.json Tilejson
TileMatrixSets
GET /tileMatrixSets Tilematrixset List
GET /tileMatrixSets/{TileMatrixSetId} Tilematrixset Info
Health Check
GET /healthz Ping


127.0.0.1:8000/public.countries/viewer
TiMVT
Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under ODbL.


127.0.0.1:8000/hexagon/viewer?step=3
TiMVT
Hudson Bay
Labrador Sea
Canada
United States of America
NORTH ATLANTIC OCEAN
Mediterranean Sea
Black Sea
Russia
Gulf of Mexico
Caribbean Sea
Red Sea
Arabian Sea
Bay of Bengal
Map tiles by Stamen Design, under CC BY 3.0. Data by OpenStreetMap, under ODbL.

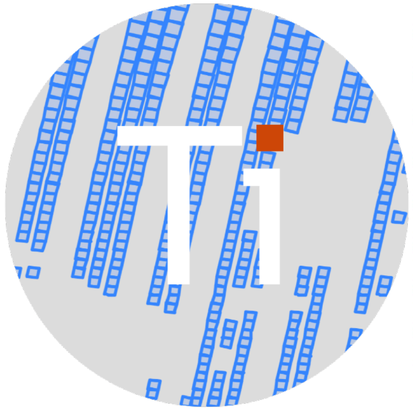
```
TIMVT_FUNCTIONS_DIRECTORY=tests/fixtures/ \
  DATABASE_URL=postgresql://username:password@0.0.0.0:5439/postgis \
  uvicorn timvt.main:app --reload

# Tables
http://127.0.0.1:8000/public.countries/viewer

# Functions
http://127.0.0.1:8000/landsat_poly_centroid/viewer

http://127.0.0.1:8000/hexagon/viewer

http://127.0.0.1:8000/hexagon/viewer?step=3

http://127.0.0.1:8000/hexagon/viewer?step=6
```

Performances

```
# Pg_TileServ
# 10 users / 100 requests
Transactions:                   1000 hits
Availability:                   100.00 %
Elapsed time:                   3.91 secs
Data transferred:               1.37 MB
Response time:                  0.04 secs
Transaction rate:               255.75 trans/sec
Throughput:                     0.35 MB/sec
Concurrency:                    9.70
Successful transactions:        1000
Failed transactions:            0
Longest transaction:            0.22
Shortest transaction:           0.00

# 20 users / 100 requests
Transactions:                   2000 hits
Availability:                   100.00 %
Elapsed time:                   6.08 secs
Data transferred:               2.74 MB
Response time:                  0.06 secs
Transaction rate:               328.95 trans/sec
Throughput:                     0.45 MB/sec
Concurrency:                    19.41
Successful transactions:        2000
Failed transactions:            0
Longest transaction:            0.25
Shortest transaction:           0.00
```

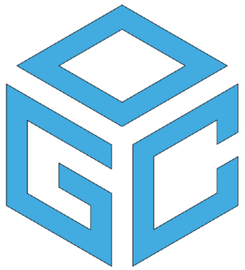
```
# Martin
# 10 users / 100 requests
Transactions:                   1000 hits
Availability:                   100.00 %
Elapsed time:                   2.46 secs
Data transferred:               1.32 MB
Response time:                  0.02 secs
Transaction rate:               406.50 trans/sec
Throughput:                     0.54 MB/sec
Concurrency:                    9.84
Successful transactions:        1000
Failed transactions:            0
Longest transaction:            0.30
Shortest transaction:           0.00

# 20 users / 100 requests
Transactions:                   2000 hits
Availability:                   100.00 %
Elapsed time:                   4.62 secs
Data transferred:               2.64 MB
Response time:                  0.05 secs
Transaction rate:               432.90 trans/sec
Throughput:                     0.57 MB/sec
Concurrency:                    19.55
Successful transactions:        2000
Failed transactions:            0
Longest transaction:            0.42
Shortest transaction:           0.00
```

```
# TiMVT (Uvicorn)
# 10 users / 100 requests
Transactions:                   1000 hits
Availability:                   100.00 %
Elapsed time:                   2.24 secs
Data transferred:               1.26 MB
Response time:                  0.02 secs
Transaction rate:               446.43 trans/sec
Throughput:                     0.56 MB/sec
Concurrency:                    9.89
Successful transactions:        1000
Failed transactions:            0
Longest transaction:            0.08
Shortest transaction:           0.01

# 20 users / 100 requests
Transactions:                   2000 hits
Availability:                   100.00 %
Elapsed time:                   4.43 secs
Data transferred:               2.52 MB
Response time:                  0.04 secs
Transaction rate:               451.47 trans/sec
Throughput:                     0.57 MB/sec
Concurrency:                    19.88
Successful transactions:        2000
Failed transactions:            0
Longest transaction:            0.38
Shortest transaction:           0.01
```



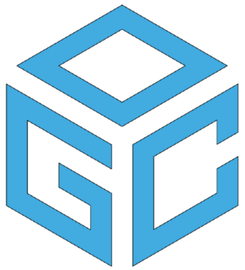
TiFeatures

https://developmentseed.org/tifeatures



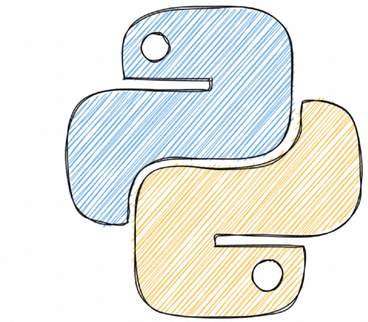
Open
Geospatial
Consortium

https://ogcapi.ogc.org/features/



Open Geospatial Consortium

Specification	Status	link
Part 1: Core	✅	docs.ogc.org/is/17-069r4/17-069r4.html
Part 2: CRS by Reference	❌	docs.ogc.org/is/18-058r1/18-058r1.html
Part 3: Filtering / CQL2	✅	docs.ogc.org/DRAFTS/19-079r1.html



python>=3.8

FastAPI

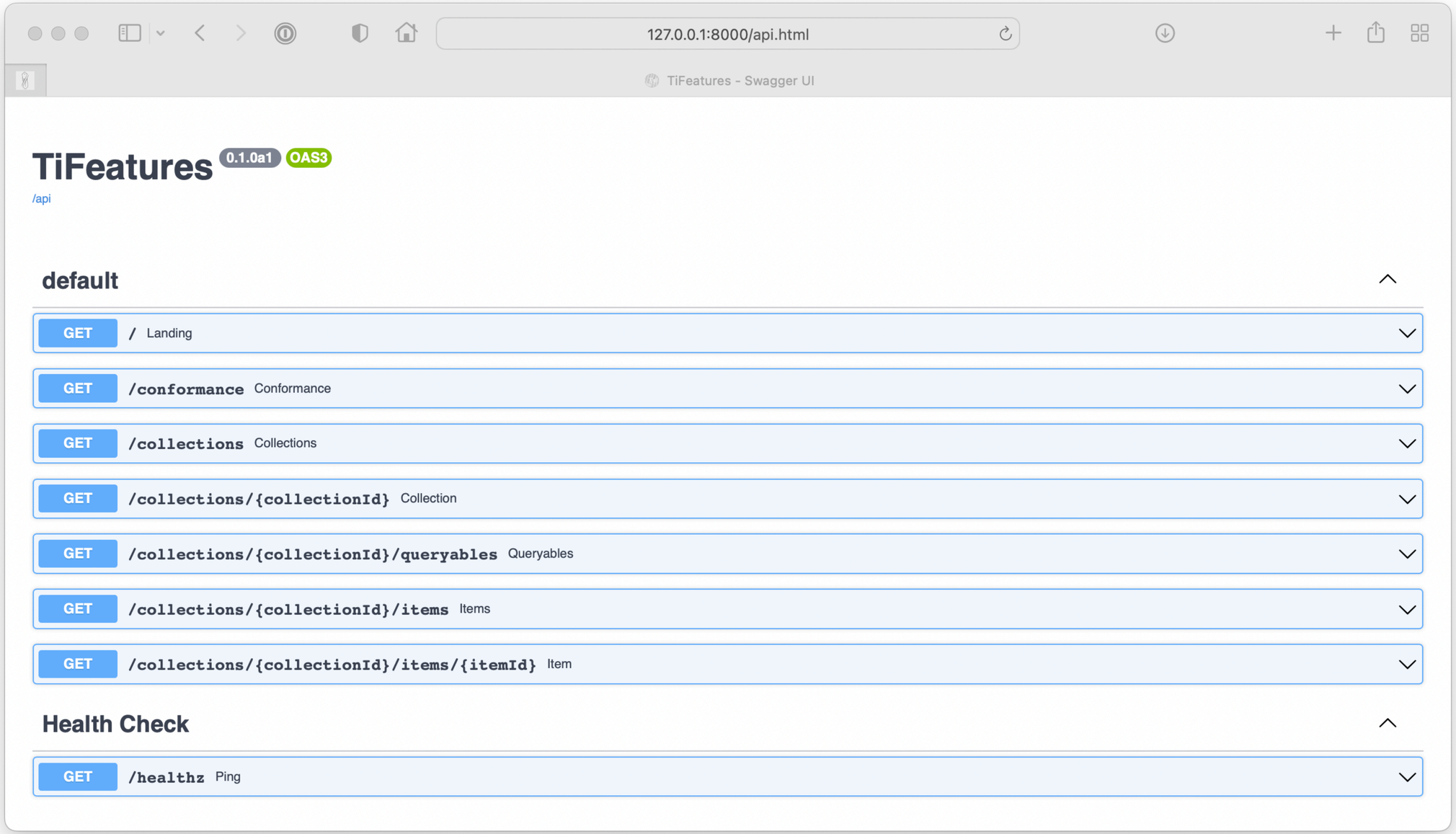

tables
Views
Functions
Database
(PostGIS)
GeoJSON
SQL
TiFeatures
geojson
html
json
csv
ndjson
geojson-seq
Async requests

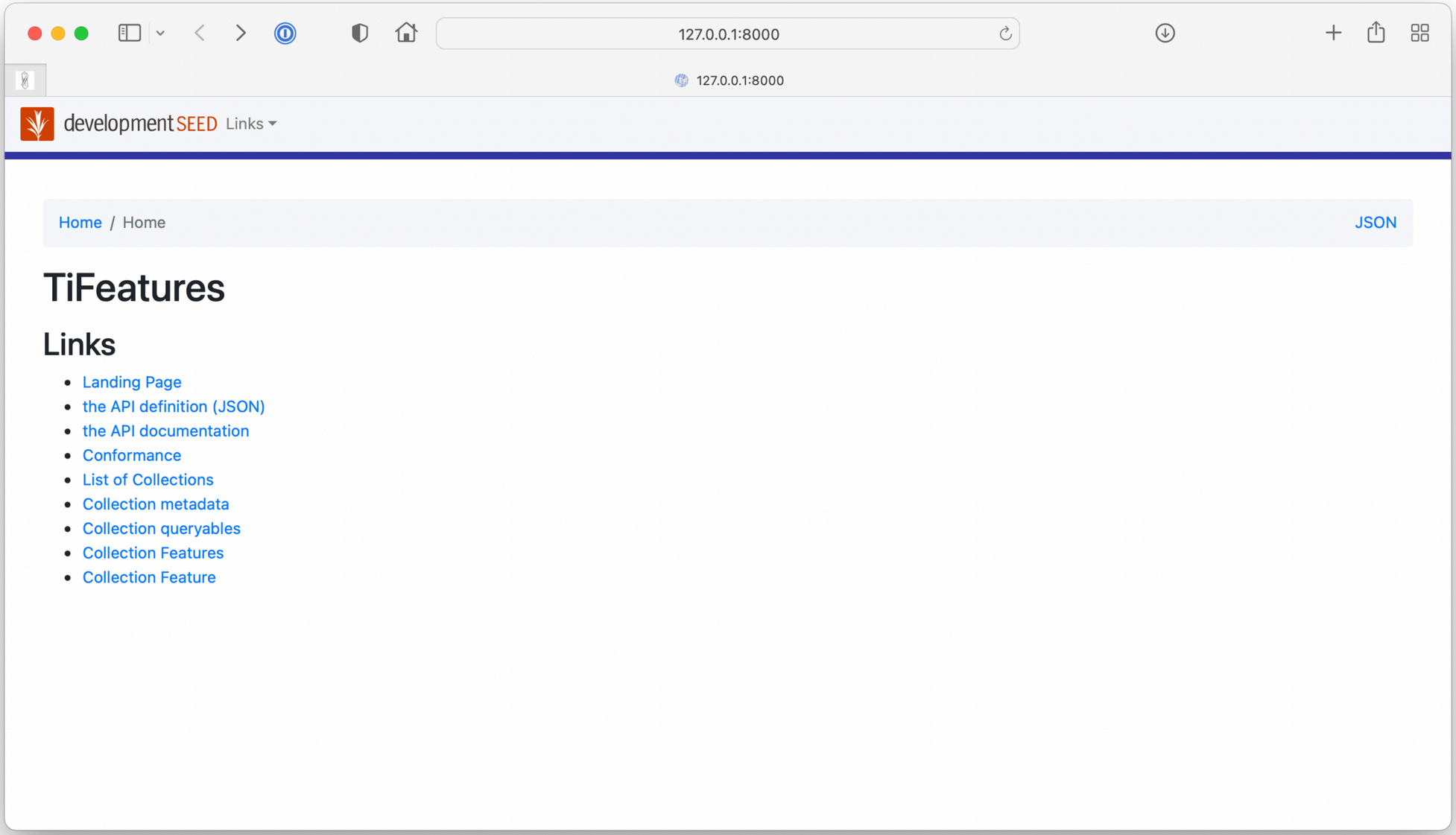
Features

- Built with FastAPI
- OGC Feature API Part 1 and Part 3
- HTML, GeoJSON, JSON, NDJSON, GeoJSON-SEQ, CSV response types
- Non-geo Table support
- Easy integration into user's application
- Async API (asyncpg)
- Table layers (Functions in Next version)
- Web Viewer
- Only need `SELECT` privileges

Don't fork it, Import it!

```
python -m pip install tifeatures
uvicorn tifeatures.main:app --port 8000
```


127.0.0.1:8000/api.html
TiFeatures - Swagger UI
TiFeatures
0.1.0a1
OAS3
/api
default
GET / Landing
GET /conformance Conformance
GET /collections Collections
GET /collections/{collectionId} Collection
GET /collections/{collectionId}/queryables Queryables
GET /collections/{collectionId}/items Items
GET /collections/{collectionId}/items/{itemId} Item
Health Check
GET /healthz Ping


127.0.0.1:8000
127.0.0.1:8000
developmentSEED Links
Home / Home
JSON
TiFeatures
Links
Landing Page
the API definition (JSON)
the API documentation
Conformance
List of Collections
Collection metadata
Collection queryables
Collection Features
Collection Feature

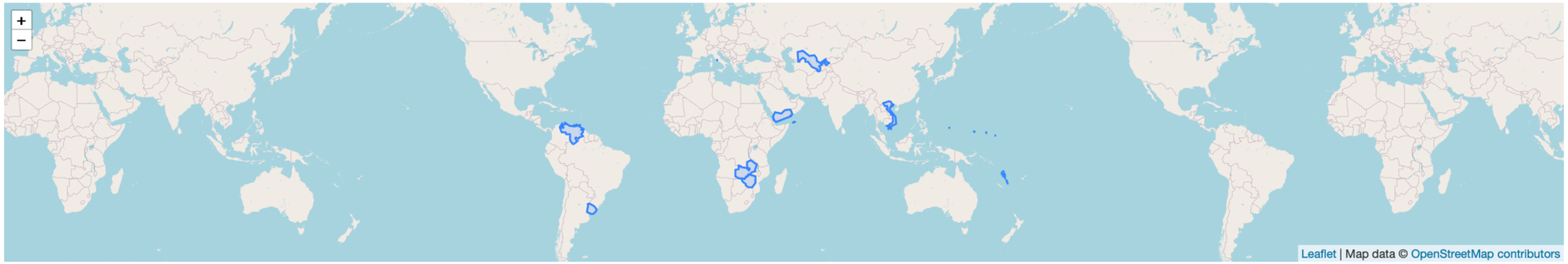
127.0.0.1:8000/collections/public.countries/items

127.0.0.1:8000/collections/public.countries/items

developmentSEED Links

Home / Collections / Public.countries / Items GeoJSON

Collection Items: public.countries



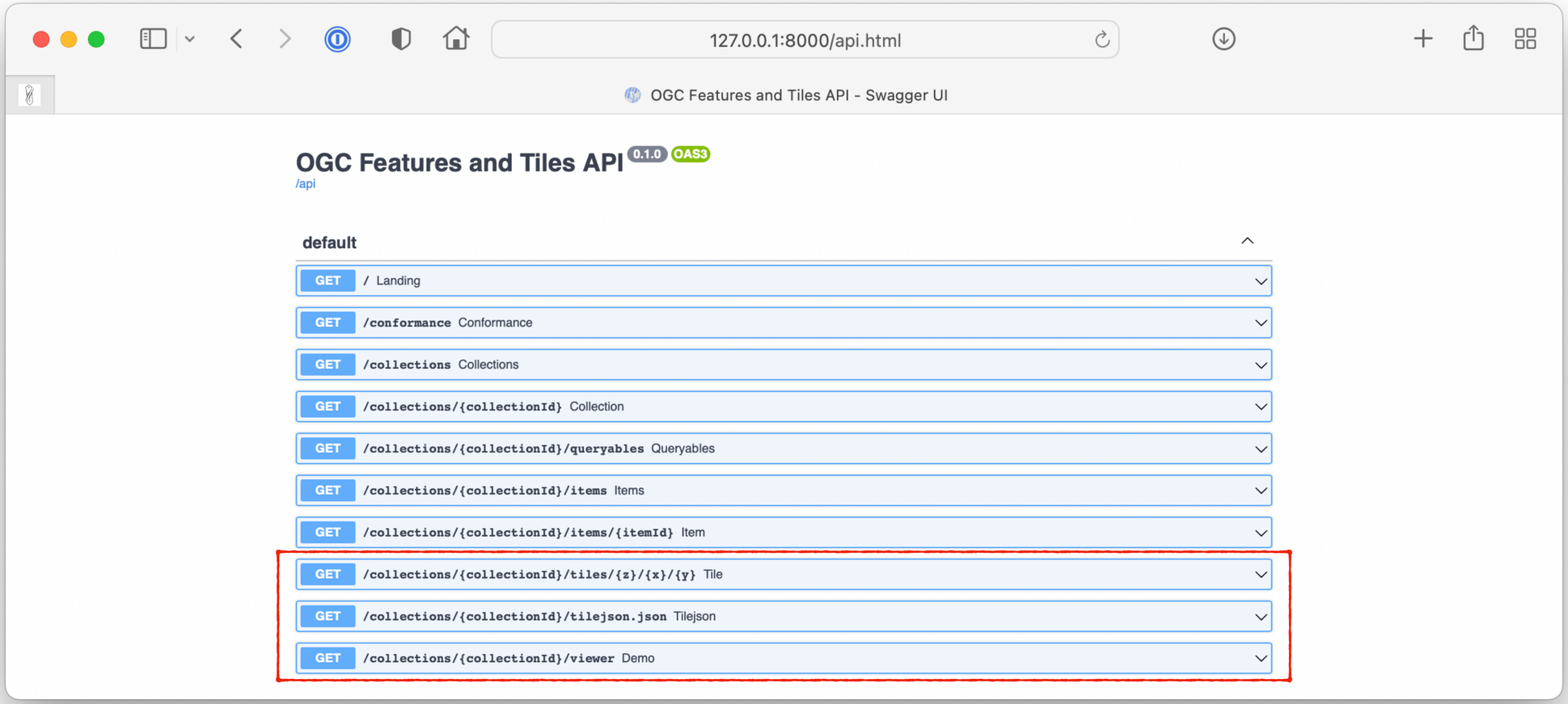
Number of matching items: 241
Number of returned items: 10
Page: 1 of 25

page size next

ID	gid	featurecla	scalerank	labelrank	sovereignt	sov_a3	adm0_dif	level	type	admin	adm0_a3	geou_dif	geounit	gu_a3	su_dif	subunit	su_a3	brk_diff	name	name_long	brk_a3	brk_name
1	1	Admin-0 country	1	3	Zimbabwe	ZWE	0	2	Sovereign country	Zimbabwe	ZWE	0	Zimbabwe	ZWE	0	Zimbabwe	ZWE	0	Zimbabwe	Zimbabwe	ZWE	Zimbabwe
2	2	Admin-0 country	1	3	Zambia	ZMB	0	2	Sovereign country	Zambia	ZMB	0	Zambia	ZMB	0	Zambia	ZMB	0	Zambia	Zambia	ZMB	Zambia
3	3	Admin-0 country	1	3	Yemen	YEM	0	2	Sovereign country	Yemen	YEM	0	Yemen	YEM	0	Yemen	YEM	0	Yemen	Yemen	YEM	Yemen
4	4	Admin-0	3	2	Vietnam	VNM	0	2	Sovereign	Vietnam	VNM	0	Vietnam	VNM	0	Vietnam	VNM	0	Vietnam	Vietnam	VNM	Vietnam

TiFeatures

```
DATABASE_URL=postgresql://username:password@0.0.0.0:5439/postgis \
  uvicorn tifeatures.main:app

# Property filter
http://127.0.0.1:8000/collections/public.countries/items?name=Canada

# Properties
http://127.0.0.1:8000/collections/public.countries/items?properties=admin,name

# CQL2 Filter
http://127.0.0.1:8000/collections/public.countries/items?sortby=-pop_est&properties=name,pop_est&filter-lang=cql2-text&filter=pop_est>40000000&limit=100

# CQL2 Filter + SQL Function
http://127.0.0.1:8000/collections/public.countries/items?filter-lang=cql2-text&filter=ST_Area(geom)>1000
```




```python
"""OGC Features and Tiles API."""

import re

from fastapi import Depends, FastAPI, HTTPException, Path, Query

from starlette.requests import Request
from starlette.responses import Response, HTMLResponse
from starlette_cramjam.middleware import CompressionMiddleware

from morecantile import Tile, tms

from tifeatures.db import close_db_connection, connect_to_db, register_table_catalog
from tifeatures.factory import Endpoints as FeaturesEndpoints

from timvt.dependencies import  TileParams
from timvt.factory import TILE_RESPONSE_PARAMS, VectorTilerFactory, templates
from timvt.layer import Layer, Table
from timvt.models.mapbox import TileJSON

app = FastAPI(
    title="OGC Features and Tiles API",
    openapi_url="/api",
    docs_url="/api.html",
)
app.add_middleware(CompressionMiddleware)


# OGC Features.
endpoints = FeaturesEndpoints()
app.include_router(endpoints.router)


# OGC Tiles
def CollectionParam(
    request: Request,
    collectionId: str = Path(..., description="collection Name"),
) -> Layer:
    """Return Layer Object."""
    table_pattern = re.match(  # type: ignore
        r"^(?P<schema>.+)\.(?P<table>.+)$", collectionId
    )
    if not table_pattern:
        raise HTTPException(status_code=404, detail=f"Invalid Table format '{collectionId}'.")

    assert table_pattern.groupdict()["schema"]
    assert table_pattern.groupdict()["table"]

    table_catalog = getattr(request.app.state, "table_catalog", {})
    if collectionId in table_catalog:
        return Table(**table_catalog[collectionId])

    raise HTTPException(status_code=404, detail=f"Table '{collectionId}' not found.")


webmercator = tms.get("WebMercatorQuad")


@app.get("/collections/{collectionId}/tiles/{z}/{x}/{y}", **TILE_RESPONSE_PARAMS)
async def tile(
    request: Request,
    tile: Tile = Depends(TileParams),
    collection=Depends(CollectionParam),
):
    """Return vector tile."""
    pool = request.app.state.pool
    content = await collection.get_tile(pool, tile, webmercator)
    return Response(bytes(content), media_type="application/x-protobuf")


@app.get(
    "/collections/{collectionId}/tilejson.json",
    response_model=TileJSON,
    responses={200: {"description": "Return a tilejson"}},
    response_model_exclude_none=True,
)
async def tilejson(request: Request, collection=Depends(CollectionParam)):
    """Return TileJSON document."""
    path_params = {
        "collectionId": collection.id,
        "z": "{z}",
        "x": "{x}",
        "y": "{y}",
    }
    tile_endpoint = request.url_for("tile", **path_params)
    return {
        "minzoom": 0,
        "maxzoom": 22,
        "name": collection.id,
        "bounds": collection.bounds,
        "tiles": [tile_endpoint],
    }


@app.get("/collections/{collectionId}/viewer", response_class=HTMLResponse)
async def demo(request: Request, collection=Depends(CollectionParam)):
    tile_url = request.url_for("tilejson", collectionId=collection.id)
    return templates.TemplateResponse(
        name="viewer.html",
        context={"endpoint": tile_url, "request": request},
        media_type="text/html",
    )


@app.on_event("startup")
async def startup_event() -> None:
    """Connect to database on startup."""
    await connect_to_db(app)
    # TiMVT and TiFeatures share the same `Table_catalog` format
    # see https://github.com/developmentseed/timvt/pull/83
    await register_table_catalog(app)


@app.on_event("shutdown")
async def shutdown_event() -> None:
    """Close database connection."""
    await close_db_connection(app)
```

```python
endpoints = FeaturesEndpoints()
app.include_router(endpoints.router)
```

OGC Features

Tiles

TileJSON

Viewer

```python
@app.get("/collections/{collectionId}/tiles/{z}/{x}/{y}", **TILE_RESPONSE_PARAMS)
async def tile(
    request: Request,
    tile: Tile = Depends(TileParams),
    collection=Depends(CollectionParam),
):
    """Return vector tile."""
    pool = request.app.state.pool
    content = await collection.get_tile(pool, tile, webmercator)
    return Response(bytes(content), media_type="application/x-protobuf")


@app.get(
    "/collections/{collectionId}/tilejson.json",
    response_model=TileJSON,
    responses={200: {"description": "Return a tilejson"}},
    response_model_exclude_none=True,
)
async def tilejson(request: Request, collection=Depends(CollectionParam)):
    """Return TileJSON document."""
    path_params = {
        "collectionId": collection.id,
        "z": "{z}",
        "x": "{x}",
        "y": "{y}",
    }
    tile_endpoint = request.url_for("tile", **path_params)
    return {
        "minzoom": 0,
        "maxzoom": 22,
        "name": collection.id,
        "bounds": collection.bounds,
        "tiles": [tile_endpoint],
    }


@app.get("/collections/{collectionId}/viewer", response_class=HTMLResponse)
async def demo(request: Request, collection=Depends(CollectionParam)):
    tile_url = request.url_for("tilejson", collectionId=collection.id)
    return templates.TemplateResponse(
        name="viewer.html",
        context={"endpoint": tile_url, "request": request},
        media_type="text/html",
    )
```

OGC Tiles

```python
"""OGC Features and Tiles API."""

import re

from fastapi import Depends, FastAPI, HTTPException, Path, Query

from starlette.requests import Request
from starlette.responses import Response, HTMLResponse
from starlette_cramjam.middleware import CompressionMiddleware

from morecantile import Tile, tms

from tifeatures.db import close_db_connection, connect_to_db, register_table_catalog
from tifeatures.factory import Endpoints as FeaturesEndpoints

from timvt.dependencies import  TileParams
from timvt.factory import TILE_RESPONSE_PARAMS, VectorTilerFactory, templates
from timvt.layer import Layer, Table
from timvt.models.mapbox import TileJSON

app = FastAPI(
    title="OGC Features and Tiles API",
    openapi_url="/api",
    docs_url="/api.html",
)
app.add_middleware(CompressionMiddleware)


# OGC Features.
endpoints = FeaturesEndpoints()
app.include_router(endpoints.router)


# OGC Tiles
def CollectionParam(
    request: Request,
    collectionId: str = Path(..., description="collection Name"),
) -> Layer:
    """Return Layer Object."""
    table_pattern = re.match(  # type: ignore
        r"^(?P<schema>.+)\.(?P<table>.+)$", collectionId
    )
    if not table_pattern:
        raise HTTPException(status_code=404, detail=f"Invalid Table format '{collectionId}'.")

    assert table_pattern.groupdict()["schema"]
    assert table_pattern.groupdict()["table"]

    table_catalog = getattr(request.app.state, "table_catalog", {})
    if collectionId in table_catalog:
        return Table(**table_catalog[collectionId])

    raise HTTPException(status_code=404, detail=f"Table '{collectionId}' not found.")


webmercator = tms.get("WebMercatorQuad")


@app.get("/collections/{collectionId}/tiles/{z}/{x}/{y}", **TILE_RESPONSE_PARAMS)
async def tile(
    request: Request,
    tile: Tile = Depends(TileParams),
    collection=Depends(CollectionParam),
):
    """Return vector tile."""
    pool = request.app.state.pool
    content = await collection.get_tile(pool, tile, webmercator)
    return Response(bytes(content), media_type="application/x-protobuf")


@app.get(
    "/collections/{collectionId}/tilejson.json",
    response_model=TileJSON,
    responses={200: {"description": "Return a tilejson"}},
    response_model_exclude_none=True,
)
async def tilejson(request: Request, collection=Depends(CollectionParam)):
    """Return TileJSON document."""
    path_params = {
        "collectionId": collection.id,
        "z": "{z}",
        "x": "{x}",
        "y": "{y}",
    }
    tile_endpoint = request.url_for("tile", **path_params)
    return {
        "minzoom": 0,
        "maxzoom": 22,
        "name": collection.id,
        "bounds": collection.bounds,
        "tiles": [tile_endpoint],
    }


@app.get("/collections/{collectionId}/viewer", response_class=HTMLResponse)
async def demo(request: Request, collection=Depends(CollectionParam)):
    tile_url = request.url_for("tilejson", collectionId=collection.id)
    return templates.TemplateResponse(
        name="viewer.html",
        context={"endpoint": tile_url, "request": request},
        media_type="text/html",
    )


@app.on_event("startup")
async def startup_event() -> None:
    """Connect to database on startup."""
    await connect_to_db(app)
    # TiMVT and TiFeatures share the same `Table_catalog` format
    # see https://github.com/developmentseed/timvt/pull/83
    await register_table_catalog(app)


@app.on_event("shutdown")
async def shutdown_event() -> None:
    """Close database connection."""
    await close_db_connection(app)
```

Performances

TiFeatures

Name (time in ms)	Min	Max	Mean	Median
test_benchmark_items_ti[geojson-1]	15.8612 (1.15)	28.4526 (1.03)	18.6286 (1.0)	17.7227 (1.0)
test_benchmark_items_ti[geojson-10]	13.8315 (1.0)	27.6663 (1.0)	19.3199 (1.04)	18.9858 (1.07)
test_benchmark_items_ti[geojson-50]	19.8409 (1.43)	42.2365 (1.53)	24.0836 (1.29)	23.2670 (1.31)
test_benchmark_items_ti[geojson-100]	20.3239 (1.47)	42.5218 (1.54)	28.1368 (1.51)	27.6908 (1.56)
test_benchmark_items_ti[geojson-200]	28.8410 (2.09)	50.3265 (1.82)	38.1986 (2.05)	37.6205 (2.12)
test_benchmark_items_ti[geojson-250]	39.6191 (2.86)	59.0787 (2.14)	44.8899 (2.41)	43.6857 (2.46)

Pg_Featureserv

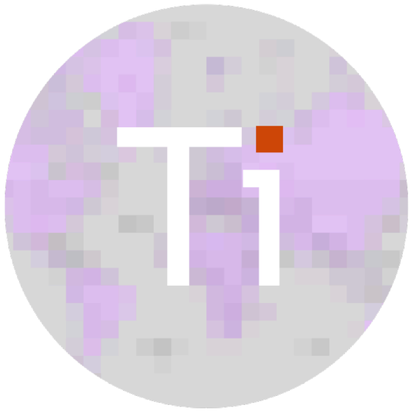
Name (time in ms)	Min	Max	Mean	Median
test_benchmark_items[json-1]	9.5992 (1.0)	20.5852 (1.0)	12.4470 (1.0)	11.6450 (1.0)
test_benchmark_items[json-10]	10.9702 (1.14)	23.0685 (1.12)	13.8784 (1.12)	13.0863 (1.12)
test_benchmark_items[json-50]	18.3388 (1.91)	28.1821 (1.37)	20.8509 (1.68)	20.1307 (1.73)
test_benchmark_items[json-100]	26.8315 (2.80)	63.4562 (3.08)	31.7796 (2.55)	29.0640 (2.50)
test_benchmark_items[json-200]	46.8038 (4.88)	58.6184 (2.85)	50.6976 (4.07)	49.4876 (4.25)
test_benchmark_items[json-250]	57.1046 (5.95)	67.3930 (3.27)	59.7314 (4.80)	59.3578 (5.10)

Ti...

python/FastAPI for everything!

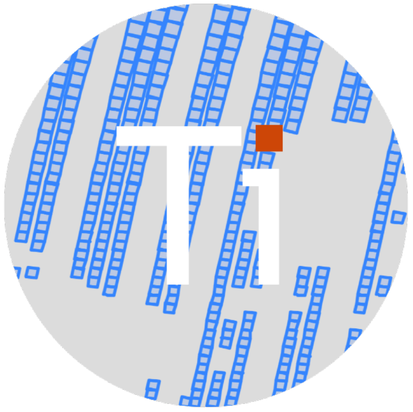
TiTiler
Raster Services

https://developmentseed.org/titiler/



TiMVT
Vector Tiles

https://developmentseed.org/timvt/



TiFeatures
OGC Features

https://developmentseed.org/tifeatures/

Next

TiFeatures + TiMVT

=

Tiles

Approved Standard

OGC API - Tiles provides extended functionality to other OGC API Standards to deliver vector tiles, map tiles, and other tiled data.

Features

Approved Standard

OGC API - Features - Part 1: Core and Part 2: Coordinate Reference Systems by Reference are both publicly available.

TiPg

Ping Me!

@_VincentS_
@cogeotiff
@developmentseed

We have 🍪

Join the team & make a better planet.
https://developmentseed.org/careers

Links

https://developmentseed.org/tifeatures/

https://developmentseed.org/timvt/

https://developmentseed.org/titiler/

https://developmentseed.org/morecantile/

https://fastapi.tiangolo.com

https://ogcapi.ogc.org/features/

https://ogcapi.ogc.org/tiles/

https://github.com/CrunchyData/pg_tileserv

https://github.com/maplibre/martin

https://github.com/CrunchyData/pg_featureserv