Focus Server – Parameterized Testing Plan

Why Parameterized Tests?

The core flows in `focus_server` (/configure, /recordings_in_time_range, gRPC streaming) repeat the same functional flow but only vary in input parameters. Instead of duplicating test code, parameterization allows us to cover broad input ranges, edge cases, and load scenarios efficiently.

Key Flows & Parameters

1. `/configure`

- **Parameters to vary**:
  - `view_type` : MULTICHANNEL, SINGLECHANNEL, WATERFALL, HEATMAP
  - `channels` : small, wide, boundary ranges (1–1, 1–8, 1–64, 32–64)
  - `nfftSelection` : 1024, 2048, 4096
  - `displayInfo.height` : 256, 600, 1024
  - `frequencyRange` : 20–3000, 80–400, invalid scenarios
  - `mode` : live (no start/end) vs historic (with start_time/end_time)
- **Expected results**:
  - Status codes: 200/202 for valid, 4xx for invalid
  - Correct derived fields: `lines_dt` , `frequencies_amount` , `stream_amount` , `channel_to_stream_index`

2. `/recordings_in_time_range`

- **Parameters to vary**:
  - Time windows: 5/10/30 min, overlaps, boundary (no recordings)
- **Expected results**:
  - Status 200/400 depending on validity
  - Proper handling of epoch conversion
  - Correct list of `[start_time, end_time]` ranges

3. Concurrency/Load

- **Parameters to vary**:
  - Parallel calls to `/configure` : 1, 10, 50
- **Expected results**:
  - Caps enforced if configured
  - `p95` time-to-ready within limits

    - No state corruption or crashes

4. gRPC streaming

- **Parameters to vary**:
    - Different `stream_id` values (0..N)
    - Keepalive config
    - Message rate variations (mocked load)
- **Expected results**:
    - Time-to-first-message within SLA
    - Expected message count
    - No unexpected errors

Example: Parameterized `/configure` (live/historic + view_type)

```python
import pytest
import requests
from datetime import datetime

BASE = "http://<focus-host>:<port>"

def build_payload(view_type, channels, nfft, height, freq_range, historic=False):
    now = int(datetime.utcnow().timestamp())
    start, end = (now - 600, now - 300) if historic else (None, None)
    return {
        "displayTimeAxisDuration": 10 if view_type != 2 else 1,
        "nfftSelection": nfft if view_type != 2 else 1,
        "displayInfo": {"height": height},
        "channels": {"min": channels[0], "max": channels[1]},
        "frequencyRange": (
            {"min": freq_range[0], "max": freq_range[1]} if view_type != 2 else None
        ),
        "start_time": start,
        "end_time": end,
        "view_type": view_type,
    }

@pytest.mark.parametrize(
    "view_type,channels,nfft,height,freq_range,historic,expected_status",
    [
        (0, (1, 8), 1024, 600, (20, 3000), False, 200),   # MULTICHANNEL live
        (0, (1, 64), 2048, 1024, (80, 400), True, 200),   # MULTICHANNEL historic
        (1, (5, 5), 1024, 256, (20, 3000), False, 200),   # SINGLECHANNEL live
        (2, (1, 32), 1, 600, None, True, 200),            # WATERFALL historic
        (3, (1, 16), 2048, 600, (80, 400), False, 200),   # HEATMAP live
        (0, (64, 1), 1024, 600, (20, 3000), False, 422),  # invalid channels
    ],
    ids=[
        "multi_live_small",
        "multi_historic_large",
        "single_live",
        "waterfall_historic",
        "heatmap_live",
        "invalid_channels"
    ],
)
def test_configure_param(view_type, channels, nfft, height, freq_range, historic, expected_status):
    body = build_payload(view_type, channels, nfft, height, freq_range, historic)
    r = requests.post(f"{BASE}/configure", json=body, timeout=30)
    assert r.status_code == expected_status
    if r.ok:
        data = r.json()
        assert "job_id" in data
        assert "stream_port" in data
        if view_type == 1:  # SINGLECHANNEL
            assert data["channel_amount"] == 1
            assert data["stream_amount"] == 1
        if view_type == 2:  # WATERFALL
            assert data["lines_dt"] > 0
        assert len(data["channel_to_stream_index"]) == data["channel_amount"]

```

Example: Parameterized `/recordings_in_time_range`

```python
import pytest
import requests
from datetime import datetime

BASE = "http://<focus-host>:<port>"

@pytest.mark.parametrize(
    "delta_start_min,delta_end_min,expected_status",
    [
        (-30, -20, 200),  # past window, likely has data
        (-10, 0, 200),    # ongoing recording
        (60, 90, 200),    # future window, may be empty but valid
    ],
    ids=["past_window","recent_window","future_window"]
)
def test_recordings_in_time_range_param(delta_start_min, delta_end_min, expected_status):
    now = int(datetime.utcnow().timestamp())
    body = {"start_time": now + delta_start_min*60, "end_time": now + delta_end_min*60}
    r = requests.post(f"{BASE}/recordings_in_time_range", json=body, timeout=15)
    assert r.status_code == expected_status
    data = r.json()
    assert isinstance(data, list)

```

Example: Concurrency Test for `/configure`

```python
import pytest, requests, concurrent.futures

BASE = "http://<focus-host>:<port>"

@pytest.mark.parametrize("concurrency", [1, 10, 50], ids=["single","x10","x50"])
def test_configure_concurrency(concurrency):
    payload = {
        "displayTimeAxisDuration": 10,
        "nfftSelection": 2048,
        "displayInfo": {"height": 600},
        "channels": {"min": 1, "max": 16},
        "frequencyRange": {"min": 80, "max": 400},
        "view_type": 0,
    }
    def call():
        return requests.post(f"{BASE}/configure", json=payload, timeout=30).status_code

    with concurrent.futures.ThreadPoolExecutor(max_workers=min(concurrency, 16)) as ex:
        results = list(ex.map(lambda _: call(), range(concurrency)))
    assert all(code in (200, 202, 429, 503) for code in results)

```

Test_configure_params_code: C:\Users\roy.avrahami\Prisma_Automation_Framework\pz\microservices\focus_server\tests\test_configure_parametrized.py