Київський політехнічний інститут імені Ігоря Сікорського

Фізико-технічний інститут

Проектування розподілених систем

Проект

Replicated log task

Виконала:

Студентка групи ФБ-42мп

Алькова Аліна

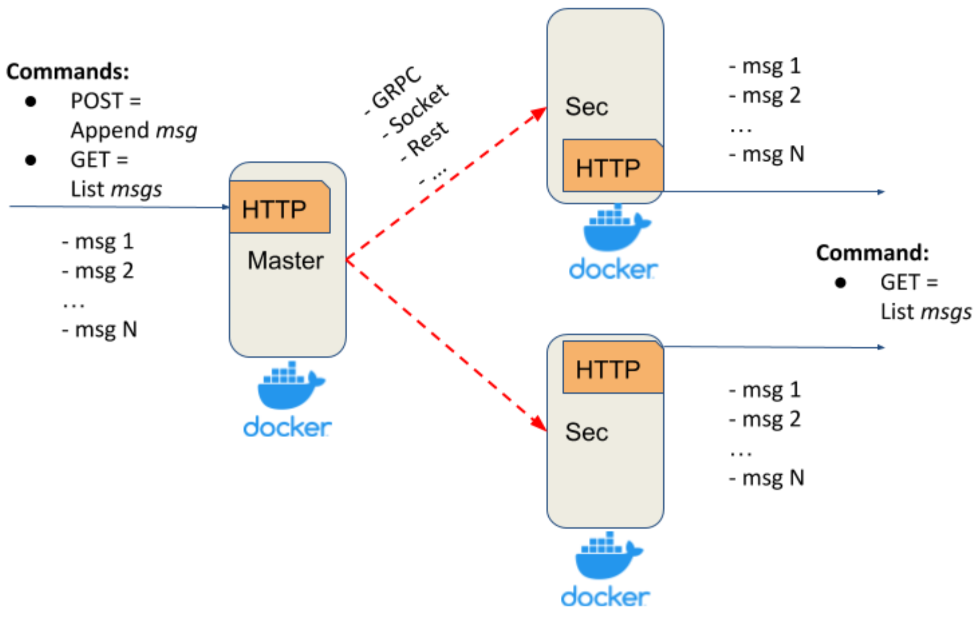
Iteration 0.

Choose a desirable language for implementation and try to implement (or find the implementation) a simple *Echo Client-Server* application.

Iteration 1.

- **5 points**

The Replicated Log should have the following deployment architecture: one ***Master*** and any number of ***Secondaries***.



Master should expose a simple HTTP server (or alternative service with a similar API) with:

- *POST method* - appends a message into the in-memory list
- *GET method* - returns all messages from the in-memory list

Secondary should expose a simple HTTP server(or alternative service with a similar API) with:

- *GET method* - returns all replicated messages from the in-memory list

Properties and assumptions:

- after each POST request, the message should be replicated on every *Secondary* server
- *Master* should ensure that *Secondaries* have received a message via *ACK*
- *Master's POST request* should be finished only after receiving *ACKs* from all *Secondaries* (blocking replication approach)
- to test that the replication is blocking, introduce the delay/sleep on the *Secondary*
- at this stage assume that the communication channel is a perfect link (no failures and messages lost)
- any RPC framework can be used for *Master-Secondary* communication (Sockets, language-specific RPC, HTTP, Rest, gRPC, …)
- your implementation should support logging
- *Master* and *Secondaries* should run in Docker

У частині я реалізовую один Master і два Secondary-сервери.

Master.py:

```
import logging
import asyncio
import os
from fastapi import FastAPI, HTTPException
import httpx
from pydantic import BaseModel
from typing import List
from fastapi.responses import JSONResponse

os.makedirs('/app/logs', exist_ok=True)

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('/app/logs/master.log'),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)

app = FastAPI()

class Message(BaseModel):
    content: str

messages: List[str] = []

SECONDARIES = ["http://secondary1:8001", "http://secondary2:8001"]

@app.post("/messages")
async def append_message(message: Message):
    logger.info(f"Received POST request with message: {message.content}")

    messages.append(message.content)

    async with httpx.AsyncClient() as client:
        for secondary in SECONDARIES:
            try:
                response = await client.post(
                    f"{secondary}/replicate",
                    json={"content": message.content},
                    timeout=10.0
                )
                if response.status_code != 200:
                    logger.error(f"Failed to replicate to {secondary}:
{response.status_code}")
                    raise HTTPException(status_code=500, detail=f"Replication
failed on {secondary}")
                logger.info(f"Successfully replicated to {secondary}")
            except httpx.RequestError as e:
                logger.error(f"Error replicating to {secondary}: {str(e)}")
                raise HTTPException(status_code=500, detail=f"Replication error
on {secondary}")

    return {"status": "Message appended and replicated", "message":
message.content}

@app.get("/messages")
async def get_messages():
    logger.info("Received GET request for messages")
```

```python
    return {"messages": messages}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

Master має HTTP API:

- POST /messages: додає повідомлення до внутрішнього списку в пам’яті.
- GET /messages: повертає всі повідомлення зі списку.

Master чекає підтвердження (ACK) від усіх Secondary перед завершенням POST-запиту (**блокуюча реплікація**). Використовується HTTP для комунікації Master-Secondary.

Secondary.py:

```python
import logging
import asyncio
import os
from fastapi import FastAPI
from pydantic import BaseModel
from typing import List

os.makedirs('/app/logs', exist_ok=True)

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('/app/logs/secondary.log'),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)

app = FastAPI()


class Message(BaseModel):
    content: str

replicated_messages: List[str] = []

@app.post("/replicate")
async def replicate_message(message: Message):
    logger.info(f"Received replication request with message: {message.content}")

    await asyncio.sleep(2)

    replicated_messages.append(message.content)

    logger.info(f"Message replicated: {message.content}")
    return {"status": "Message replicated"}

@app.get("/messages")
async def get_messages():
    logger.info("Received GET request for replicated messages")
    return {"messages": replicated_messages}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8001)
```

Secondary-сервери:

- Ендпоінт POST /replicate: Приймає повідомлення від Master і додає їх до локального списку з затримкою 2 секунди для тестування блокуючої реплікації.
- Ендпоінт GET /messages: Повертає список реплікованих повідомлень.

Також для перевірки роботи системи я пишу тести:

```python
import pytest
import httpx
import asyncio
from time import time

MASTER_URL = "http://localhost:8000"
SECONDARY1_URL = "http://localhost:8001"
SECONDARY2_URL = "http://localhost:8002"

@pytest.mark.asyncio
async def test_append_and_replicate():
    async with httpx.AsyncClient() as client:
        start_time = time()
        response = await client.post(f"{MASTER_URL}/messages", json={"content": "Test message"})
        end_time = time()

        assert response.status_code == 200
        assert response.json()["message"] == "Test message"
        assert end_time - start_time >= 4.0

        master_response = await client.get(f"{MASTER_URL}/messages")
        assert master_response.status_code == 200
        assert "Test message" in master_response.json()["messages"]

        for secondary_url in [SECONDARY1_URL, SECONDARY2_URL]:
            secondary_response = await client.get(f"{secondary_url}/messages")
            assert secondary_response.status_code == 200
            assert "Test message" in secondary_response.json()["messages"]

@pytest.mark.asyncio
async def test_get_empty_messages():
    async with httpx.AsyncClient() as client:
        response = await client.get(f"{MASTER_URL}/messages")
        assert response.status_code == 200
        assert isinstance(response.json()["messages"], list)

        for secondary_url in [SECONDARY1_URL, SECONDARY2_URL]:
            response = await client.get(f"{secondary_url}/messages")
            assert response.status_code == 200
            assert isinstance(response.json()["messages"], list)
```

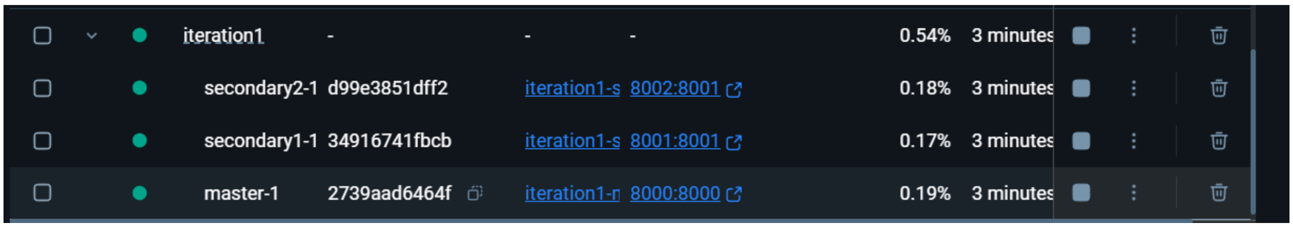
test_append_and_replicate: перевіряє, що повідомлення, надіслане через POST-запит до Master, додається до його списку і реплікується на всі Secondary-сервери, а також що реплікація є блокуючою.

- Відправляє POST-запит до http://localhost:8000/messages із повідомленням "Test message".
- Перевіряє, що:
  - Запит повертає статус 200 і коректне повідомлення у відповіді ({"message": "Test message"}).

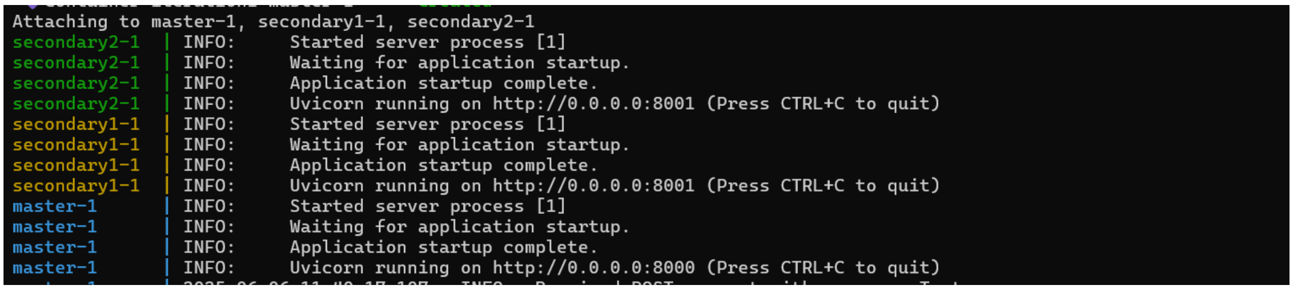
- Виконання запиту займає щонайменше 4 секунди (затримка 2 секунди на кожному Secondary, що підтверджує блокуючу реплікацію).
- Повідомлення "Test message" з'являється в списку Master (через GET-запит до http://localhost:8000/messages).
- Повідомлення "Test message" репліковано на обидва Secondary-сервери (через GET-запити до http://localhost:8001/messages і http://localhost:8002/messages).

test_get_empty_messages: перевіряє, що GET-запити до Master і Secondary повертають коректний список повідомлень (навіть якщо список порожній або містить дані). Він відправляє GET-запити до http://localhost:8000/messages, http://localhost:8001/messages і http://localhost:8002/messages.

Запуск:

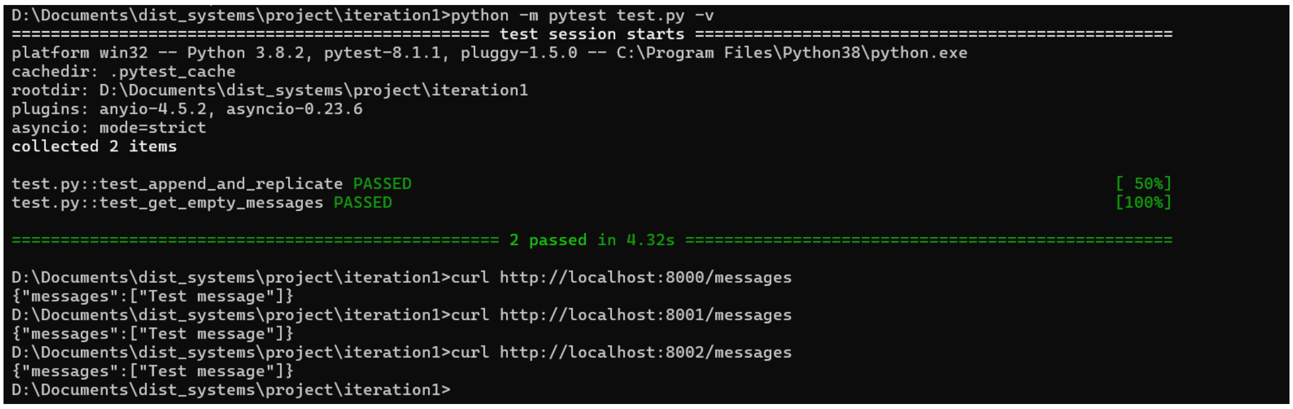


☐	⌄	●	iteration1	-	-	-	0.54%	3 minutes
☐		●	secondary2-1	d99e3851dff2	iteration1-s	8002:8001	0.18%	3 minutes
☐		●	secondary1-1	34916741fbcb	iteration1-s	8001:8001	0.17%	3 minutes
☐		●	master-1	2739aad6464f	iteration1-m	8000:8000	0.19%	3 minutes



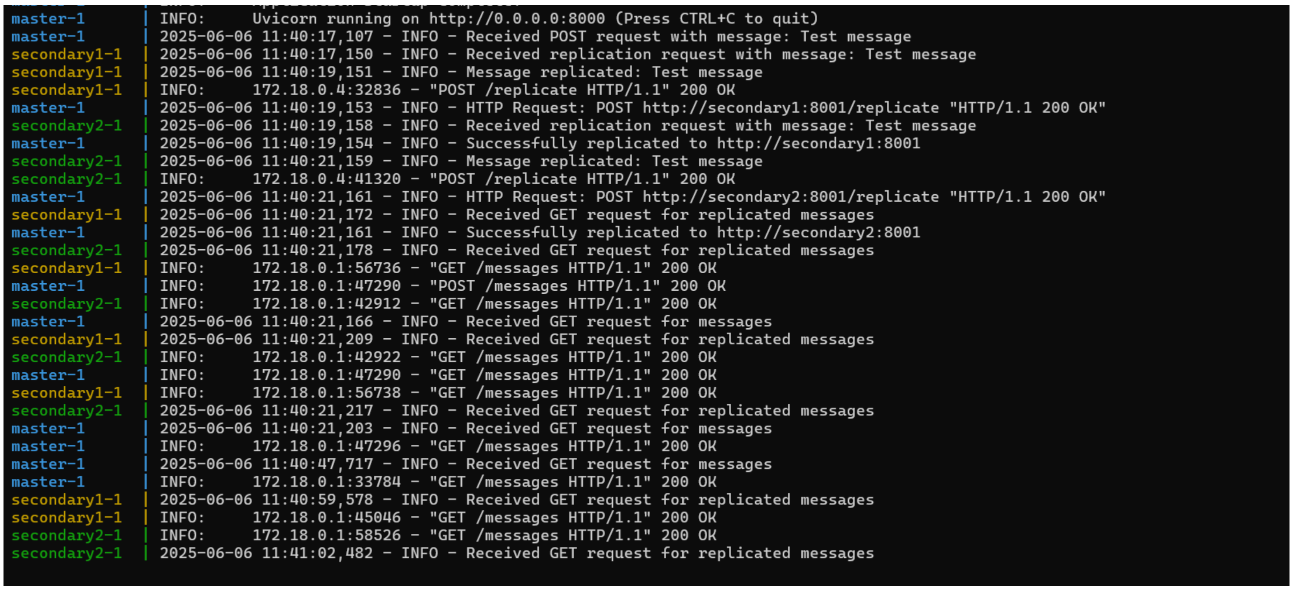
```
Attaching to master-1, secondary1-1, secondary2-1
secondary2-1  | INFO:     Started server process [1]
secondary2-1  | INFO:     Waiting for application startup.
secondary2-1  | INFO:     Application startup complete.
secondary2-1  | INFO:     Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
secondary1-1  | INFO:     Started server process [1]
secondary1-1  | INFO:     Waiting for application startup.
secondary1-1  | INFO:     Application startup complete.
secondary1-1  | INFO:     Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
master-1      | INFO:     Started server process [1]
master-1      | INFO:     Waiting for application startup.
master-1      | INFO:     Application startup complete.
master-1      | INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

Перевірка тестів:



```
D:\Documents\dist_systems\project\iteration1>python -m pytest test.py -v
================================================ test session starts ================================================
platform win32 -- Python 3.8.2, pytest-8.1.1, pluggy-1.5.0 -- C:\Program Files\Python38\python.exe
cachedir: .pytest_cache
rootdir: D:\Documents\dist_systems\project\iteration1
plugins: anyio-4.5.2, asyncio-0.23.6
asyncio: mode=strict
collected 2 items

test.py::test_append_and_replicate PASSED                                                                   [ 50%]
test.py::test_get_empty_messages PASSED                                                                     [100%]

================================================= 2 passed in 4.32s =================================================

D:\Documents\dist_systems\project\iteration1>curl http://localhost:8000/messages
{"messages":["Test message"]}
D:\Documents\dist_systems\project\iteration1>curl http://localhost:8001/messages
{"messages":["Test message"]}
D:\Documents\dist_systems\project\iteration1>curl http://localhost:8002/messages
{"messages":["Test message"]}
D:\Documents\dist_systems\project\iteration1>
```

Логи:



```
master-1       | INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
master-1       | 2025-06-06 11:40:17,107 - INFO - Received POST request with message: Test message
secondary1-1   | 2025-06-06 11:40:17,150 - INFO - Received replication request with message: Test message
secondary1-1   | 2025-06-06 11:40:19,151 - INFO - Message replicated: Test message
secondary1-1   | INFO:     172.18.0.4:32836 - "POST /replicate HTTP/1.1" 200 OK
master-1       | 2025-06-06 11:40:19,153 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
secondary2-1   | 2025-06-06 11:40:19,158 - INFO - Received replication request with message: Test message
master-1       | 2025-06-06 11:40:19,154 - INFO - Successfully replicated to http://secondary1:8001
secondary2-1   | 2025-06-06 11:40:21,159 - INFO - Message replicated: Test message
secondary2-1   | INFO:     172.18.0.4:41320 - "POST /replicate HTTP/1.1" 200 OK
master-1       | 2025-06-06 11:40:21,161 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
secondary1-1   | 2025-06-06 11:40:21,172 - INFO - Received GET request for replicated messages
master-1       | 2025-06-06 11:40:21,161 - INFO - Successfully replicated to http://secondary2:8001
secondary2-1   | 2025-06-06 11:40:21,178 - INFO - Received GET request for replicated messages
secondary1-1   | INFO:     172.18.0.1:56736 - "GET /messages HTTP/1.1" 200 OK
master-1       | INFO:     172.18.0.1:47290 - "POST /messages HTTP/1.1" 200 OK
secondary2-1   | INFO:     172.18.0.1:42912 - "GET /messages HTTP/1.1" 200 OK
master-1       | 2025-06-06 11:40:21,166 - INFO - Received GET request for messages
secondary1-1   | 2025-06-06 11:40:21,209 - INFO - Received GET request for replicated messages
secondary2-1   | INFO:     172.18.0.1:42922 - "GET /messages HTTP/1.1" 200 OK
master-1       | INFO:     172.18.0.1:47290 - "GET /messages HTTP/1.1" 200 OK
secondary1-1   | INFO:     172.18.0.1:56738 - "GET /messages HTTP/1.1" 200 OK
secondary2-1   | 2025-06-06 11:40:21,217 - INFO - Received GET request for replicated messages
master-1       | 2025-06-06 11:40:21,203 - INFO - Received GET request for messages
master-1       | INFO:     172.18.0.1:47296 - "GET /messages HTTP/1.1" 200 OK
master-1       | 2025-06-06 11:40:47,717 - INFO - Received GET request for messages
master-1       | INFO:     172.18.0.1:33784 - "GET /messages HTTP/1.1" 200 OK
secondary1-1   | 2025-06-06 11:40:59,578 - INFO - Received GET request for replicated messages
secondary1-1   | INFO:     172.18.0.1:45046 - "GET /messages HTTP/1.1" 200 OK
secondary2-1   | INFO:     172.18.0.1:58526 - "GET /messages HTTP/1.1" 200 OK
secondary2-1   | 2025-06-06 11:41:02,482 - INFO - Received GET request for replicated messages
```

Обидва тести пройшли успішно, що підтверджує коректність роботи системи.

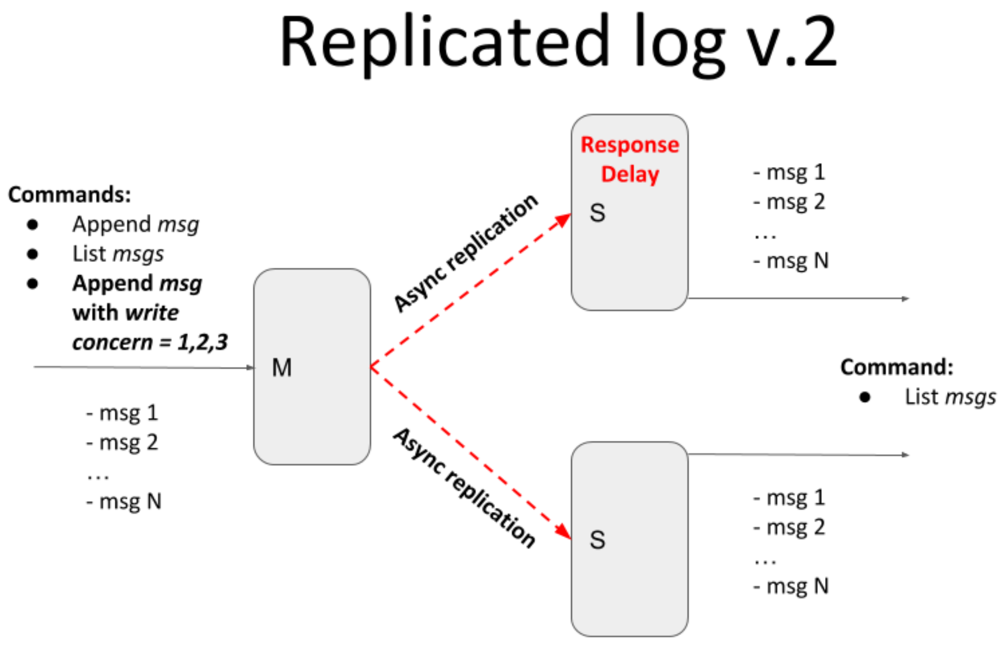
Запити curl до http://localhost:8000/messages, http://localhost:8001/messages і http://localhost:8002/messages повернули {"messages":"Test message"} = успішне збереження і реплікація повідомлення.

Логи контейнерів показують коректну обробку POST і GET запитів, а також затримку в 2 секунди на Secondary-серверах = блокуюча реплікація.

Iteration 2.

- **5 points**

In the previous iteration, the replication was blocking for all secondaries, i.e. to return a response to the client we should receive acknowledgements (ACK) from all secondaries.



Current iteration should provide tunable semi-synchronicity for replication, by defining *write concern* parameters.

- client POST request in addition to the message should also contain *write concern* parameter *w=1,2,3,..,n*
- *w* value specifies how many ACKs the master should receive from secondaries before responding to the client
  w = 1 - only from master
  w = 2 - from master and one secondary
  w = 3 - from master and two secondaries

Please emulate the replica’s inconsistency (and eventual consistency) with the master by introducing the artificial delay on the secondary node. In this case, the master and secondary should temporarily return different lists of messages.
Add logic for messages deduplication and to guarantee the total ordering of messages.

Тут я знову реалізовую **master.py**, який приймає клієнтські запити (POST для додавання повідомлень, GET для отримання списку повідомлень) і координує реплікацію на Secondary-сервери:

```
import logging
import asyncio
import os
import uuid
from fastapi import FastAPI, HTTPException
import httpx
from pydantic import BaseModel
from typing import List
from fastapi.responses import JSONResponse
```

```python
os.makedirs('/app/logs', exist_ok=True)

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('/app/logs/master.log'),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)

app = FastAPI()

class Message(BaseModel):
    content: str
    w: int

class StoredMessage(BaseModel):
    message_id: str
    content: str

messages: List[StoredMessage] = []

SECONDARIES = ["http://secondary1:8001", "http://secondary2:8001"]

@app.post("/messages")
async def append_message(message: Message):
    if message.w < 1 or message.w > len(SECONDARIES) + 1:
        logger.error(f"Invalid write concern: w={message.w}")
        raise HTTPException(status_code=400, detail=f"Write concern must be
between 1 and {len(SECONDARIES) + 1}")

    message_id = str(uuid.uuid4())
    if any(m.content == message.content for m in messages):
        logger.info(f"Message with content: {message.content} already exists,
returning existing message_id")
        existing_message = next(m for m in messages if m.content ==
message.content)
        return {"status": "Message already exists", "message_id":
existing_message.message_id, "content": message.content}

    logger.info(f"Received POST request with message_id: {message_id}, content:
{message.content}, w: {message.w}")

    messages.append(StoredMessage(message_id=message_id,
content=message.content))

    async def replicate_to_secondary(secondary: str):
        async with httpx.AsyncClient() as client:
            try:
                response = await client.post(
                    f"{secondary}/replicate",
                    json={"message_id": message_id, "content": message.content},
                    timeout=10.0
                )
                if response.status_code == 200:
                    logger.info(f"Successfully replicated to {secondary}")
                    return True
                else:
                    logger.error(f"Failed to replicate to {secondary}:
{response.status_code}")
                    return False
            except httpx.RequestError as e:
                logger.error(f"Error replicating to {secondary}: {str(e)}")
                return False
    successful_acks = 1
    for i in range(min(message.w - 1, len(SECONDARIES))):
        if await replicate_to_secondary(SECONDARIES[i]):
            successful_acks += 1
        else:
            logger.error(f"Failed to get ACK from {SECONDARIES[i]}")
            raise HTTPException(status_code=500, detail=f"Failed to replicate to
{SECONDARIES[i]}")

    if successful_acks < message.w:
        logger.error(f"Not enough ACKs: got {successful_acks}, required
{message.w}")
        raise HTTPException(status_code=500, detail="Not enough ACKs from
secondaries")

    if message.w - 1 < len(SECONDARIES):
        for secondary in SECONDARIES[message.w - 1:]:
            asyncio.create_task(replicate_to_secondary(secondary))

    return {"status": "Message appended and replicated", "message_id":
message_id, "content": message.content}

@app.get("/messages")
async def get_messages():
    logger.info("Received GET request for messages")
    return {"messages": [{"message_id": m.message_id, "content": m.content} for
m in messages]}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

- POST /messages: Додає нове повідомлення з параметром w (write concern), перевіряє унікальність (content), генерує message_id, відправляє на w-1 Secondary послідовно і на решту асинхронно.
- GET /messages: Повертає список усіх повідомлень із message_id і
- Використовує FastAPI для HTTP API.
- Використовує uuid для генерації унікальних message_id.

Далі **secondary.py:**

```
import logging
import asyncio
import os
from fastapi import FastAPI
from pydantic import BaseModel
from typing import List

os.makedirs('/app/logs', exist_ok=True)

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('/app/logs/secondary.log'),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)

app = FastAPI()
```

```
class Message(BaseModel):
    message_id: str
    content: str

replicated_messages: List[Message] = []

@app.post("/replicate")
async def replicate_message(message: Message):
    logger.info(f"Received replication request with message_id:
{message.message_id}, content: {message.content}")

    if any(m.message_id == message.message_id for m in replicated_messages):
        logger.info(f"Message with message_id: {message.message_id} already
exists, skipping")
        return {"status": "Message already replicated"}

    await asyncio.sleep(2)
    replicated_messages.append(message)

    logger.info(f"Message replicated: {message.message_id}, {message.content}")
    return {"status": "Message replicated"}

@app.get("/messages")
async def get_messages():
    logger.info("Received GET request for replicated messages")
    return {"messages": [{"message_id": m.message_id, "content": m.content} for
m in replicated_messages]}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8001)
```

Secondary-сервер отримує реплікаційні запити від Master і зберігає повідомлення.

- POST /replicate: Отримує повідомлення від Master, перевіряє унікальність за message_id, додає із затримкою 2 секунди (для імітації неконсистентності).
- GET /messages: Повертає список реплікованих повідомлень.

Для цієї частини я також пишу тести для перевірки усіх вимог:

```
import pytest
import httpx
import asyncio
from time import time

MASTER_URL = "http://localhost:8000"
SECONDARY1_URL = "http://localhost:8001"
SECONDARY2_URL = "http://localhost:8002"

@pytest.mark.asyncio
async def test_write_concern_1():
    async with httpx.AsyncClient() as client:
        start_time = time()
        response = await client.post(f"{MASTER_URL}/messages", json={"content":
"Test w=1", "w": 1})
        end_time = time()

        assert response.status_code == 200
        assert response.json()["content"] == "Test w=1"
        assert end_time - start_time < 1.0
        master_response = await client.get(f"{MASTER_URL}/messages")
        assert master_response.status_code == 200
        assert any(m["content"] == "Test w=1" for m in
master_response.json()["messages"])
        await asyncio.sleep(3)
        for secondary_url in [SECONDARY1_URL, SECONDARY2_URL]:
            secondary_response = await client.get(f"{secondary_url}/messages")
            assert secondary_response.status_code == 200
            assert any(m["content"] == "Test w=1" for m in
secondary_response.json()["messages"])

@pytest.mark.asyncio
async def test_write_concern_2():
    async with httpx.AsyncClient() as client:
        start_time = time()
        response = await client.post(f"{MASTER_URL}/messages", json={"content":
"Test w=2", "w": 2})
        end_time = time()

        assert response.status_code == 200
        assert response.json()["content"] == "Test w=2"
        assert 2.0 <= end_time - start_time < 3.0

        master_response = await client.get(f"{MASTER_URL}/messages")
        assert master_response.status_code == 200
        assert any(m["content"] == "Test w=2" for m in
master_response.json()["messages"])

        secondary1_response = await client.get(SECONDARY1_URL + "/messages")
        assert secondary1_response.status_code == 200
        assert any(m["content"] == "Test w=2" for m in
secondary1_response.json()["messages"])

        secondary2_response = await client.get(SECONDARY2_URL + "/messages")
        assert secondary2_response.status_code == 200
        has_message = any(m["content"] == "Test w=2" for m in
secondary2_response.json()["messages"])
        if not has_message:
            await asyncio.sleep(3)
            secondary2_response = await client.get(SECONDARY2_URL + "/messages")
            assert any(m["content"] == "Test w=2" for m in
secondary2_response.json()["messages"])

@pytest.mark.asyncio
async def test_write_concern_3():
    async with httpx.AsyncClient() as client:
        start_time = time()
        response = await client.post(f"{MASTER_URL}/messages", json={"content":
"Test w=3", "w": 3})
        end_time = time()

        assert response.status_code == 200
        assert response.json()["content"] == "Test w=3"
        assert end_time - start_time >= 4.0

        master_response = await client.get(f"{MASTER_URL}/messages")
        assert master_response.status_code == 200
        assert any(m["content"] == "Test w=3" for m in
master_response.json()["messages"])

        for secondary_url in [SECONDARY1_URL, SECONDARY2_URL]:
            secondary_response = await client.get(f"{secondary_url}/messages")
            assert secondary_response.status_code == 200
            assert any(m["content"] == "Test w=3" for m in
secondary_response.json()["messages"])

@pytest.mark.asyncio
async def test_deduplication():
    async with httpx.AsyncClient() as client:
        response1 = await client.post(f"{MASTER_URL}/messages", json={"content":
"Duplicate test", "w": 3})
```

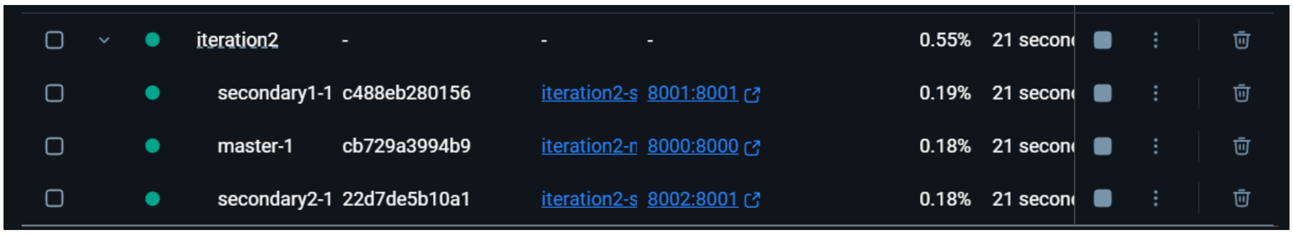
```
        response2 = await client.post(f"{MASTER_URL}/messages", json={"content":
"Duplicate test", "w": 3})

        master_response = await client.get(f"{MASTER_URL}/messages")
        assert master_response.status_code == 200
        duplicates = sum(1 for m in master_response.json()["messages"] if
m["content"] == "Duplicate test")
        assert duplicates == 1

        await asyncio.sleep(3)
        for secondary_url in [SECONDARY1_URL, SECONDARY2_URL]:
            secondary_response = await client.get(f"{secondary_url}/messages")
            assert secondary_response.status_code == 200
            duplicates = sum(1 for m in secondary_response.json()["messages"] if
m["content"] == "Duplicate test")
            assert duplicates == 1

@pytest.mark.asyncio
async def test_total_ordering():
    async with httpx.AsyncClient() as client:
        messages = ["First", "Second", "Third"]
        for msg in messages:
            await client.post(f"{MASTER_URL}/messages", json={"content": msg,
"w": 3})
        await asyncio.sleep(3)

        master_response = await client.get(f"{MASTER_URL}/messages")
        assert master_response.status_code == 200
        master_messages = [m["content"] for m in
master_response.json()["messages"]][-3:]
        assert master_messages == messages

        for secondary_url in [SECONDARY1_URL, SECONDARY2_URL]:
            secondary_response = await client.get(f"{secondary_url}/messages")
            assert secondary_response.status_code == 200
            secondary_messages = [m["content"] for m in
secondary_response.json()["messages"]][-3:]
            assert secondary_messages == messages
```

1. **test_write_concern_1** - перевіряє, що для w=1 Master додає повідомлення без очікування Secondary і що повідомлення з'являються на Secondary асинхронно.
   - **Логіка**:
     - Відправляє POST-запит до Master із content="Test w=1", w=1.
     - Перевіряє:
       - Статус-код 200.
       - Повернене повідомлення містить "Test w=1".
       - Час виконання < 1 секунди (бо не чекаємо Secondary).
     - Отримує список повідомлень із Master (GET /messages) і перевіряє наявність "Test w=1".
     - Чекає 3 секунди (щоб Secondary завершили реплікацію) і перевіряє, що "Test w=1" є на обох Secondary.
2. **test_write_concern_2** - перевіряє, що для w=2 Master чекає підтвердження від одного Secondary і демонструє неконсистентність (другий Secondary може не мати повідомлення одразу).
   - **Логіка**:
     - Відправляє POST із content="Test w=2", w=2.
     - Перевіряє:
       - Статус 200.
       - Час виконання ~2-3 секунди (2 секунди затримки від одного Secondary).

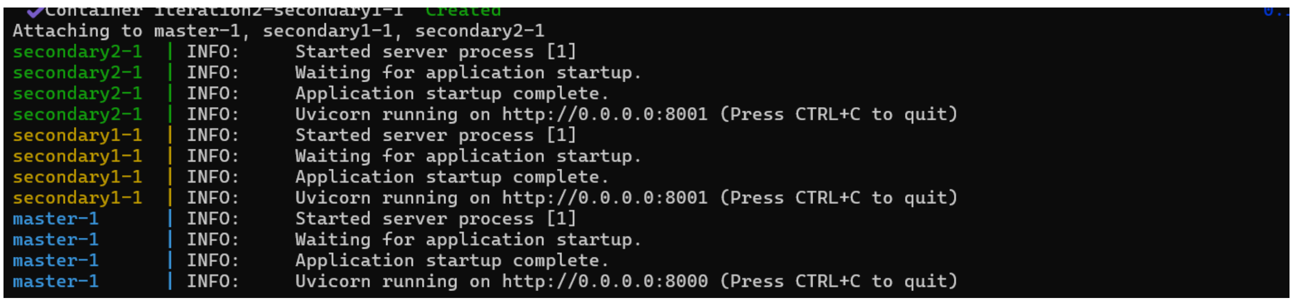
        ▪ Перевіряє, що "Test w=2" є на Master і Secondary1.
        ▪ Перевіряє, що на Secondary2 повідомлення може бути відсутнє одразу, але з’явиться після 3 секунд.

3. **test_write_concern_3** - перевіряє, що для w=3 Master чекає підтвердження від обох Secondary.
4. **test_deduplication** - перевіряє, що однакові повідомлення (content) додаються лише раз. Відправляє два POST-запити з однаковим content="Duplicate test", w=3. Перевіряє, що на Master і Secondary є лише одне повідомлення "Duplicate test".
5. **test_total_ordering** - Відправляє три POST-запити з content="First", "Second", "Third", w=3. Чекає 3 секунди і перевіряє, що порядок повідомлень однаковий на Master і обох Secondary.

Запуск:

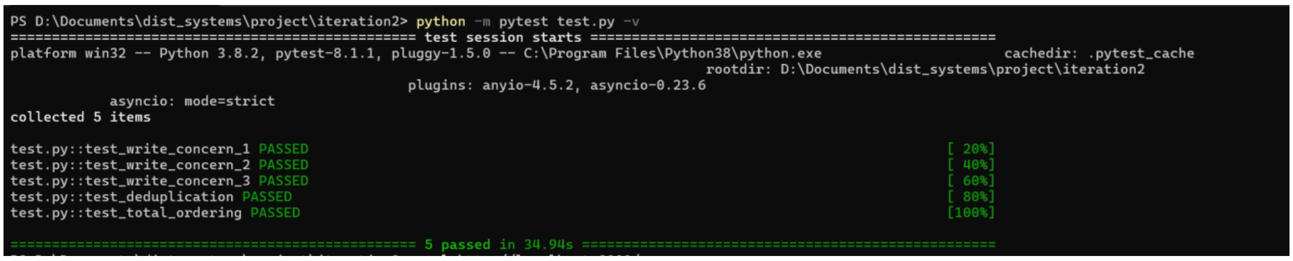


iteration2	-	-	-	0.55%	21 secon
secondary1-1	c488eb280156	iteration2-s	8001:8001	0.19%	21 secon
master-1	cb729a3994b9	iteration2-n	8000:8000	0.18%	21 secon
secondary2-1	22d7de5b10a1	iteration2-s	8002:8001	0.18%	21 secon



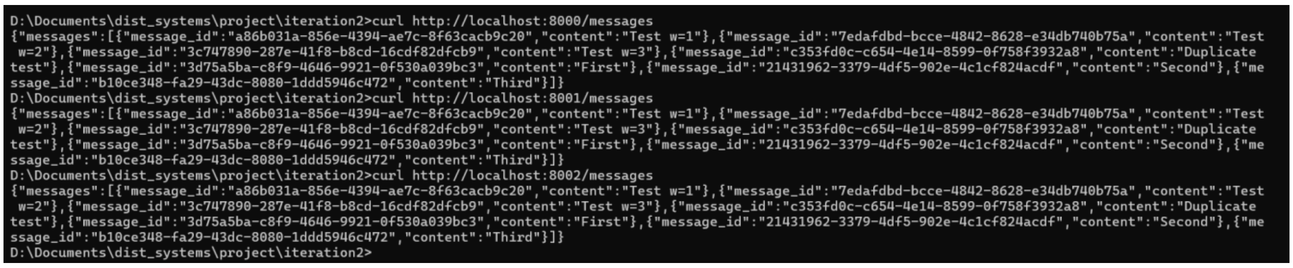
```
 ✔Container iteration2-secondary1-1  Created
Attaching to master-1, secondary1-1, secondary2-1
secondary2-1  | INFO:     Started server process [1]
secondary2-1  | INFO:     Waiting for application startup.
secondary2-1  | INFO:     Application startup complete.
secondary2-1  | INFO:     Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
secondary1-1  | INFO:     Started server process [1]
secondary1-1  | INFO:     Waiting for application startup.
secondary1-1  | INFO:     Application startup complete.
secondary1-1  | INFO:     Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
master-1      | INFO:     Started server process [1]
master-1      | INFO:     Waiting for application startup.
master-1      | INFO:     Application startup complete.
master-1      | INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

Перевірка тестів:

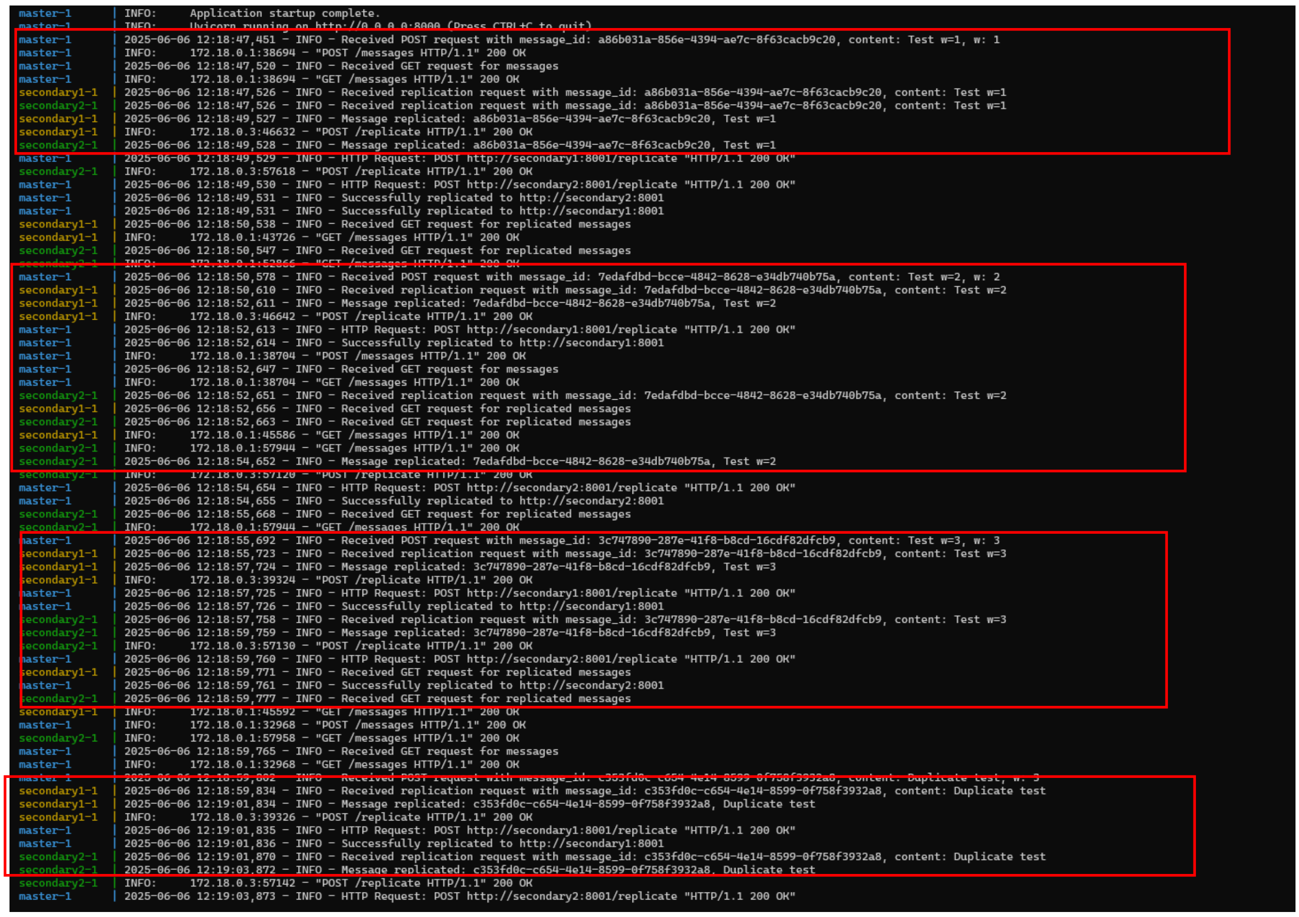


```
PS D:\Documents\dist_systems\project\iteration2> python -m pytest test.py -v
================================================= test session starts =================================================
platform win32 -- Python 3.8.2, pytest-8.1.1, pluggy-1.5.0 -- C:\Program Files\Python38\python.exe                      cachedir: .pytest_cache
                                                                                     rootdir: D:\Documents\dist_systems\project\iteration2
                                                    plugins: anyio-4.5.2, asyncio-0.23.6
            asyncio: mode=strict
collected 5 items

test.py::test_write_concern_1 PASSED                                                                                   [ 20%]
test.py::test_write_concern_2 PASSED                                                                                   [ 40%]
test.py::test_write_concern_3 PASSED                                                                                   [ 60%]
test.py::test_deduplication PASSED                                                                                     [ 80%]
test.py::test_total_ordering PASSED                                                                                    [100%]

================================================== 5 passed in 34.94s ==================================================
```

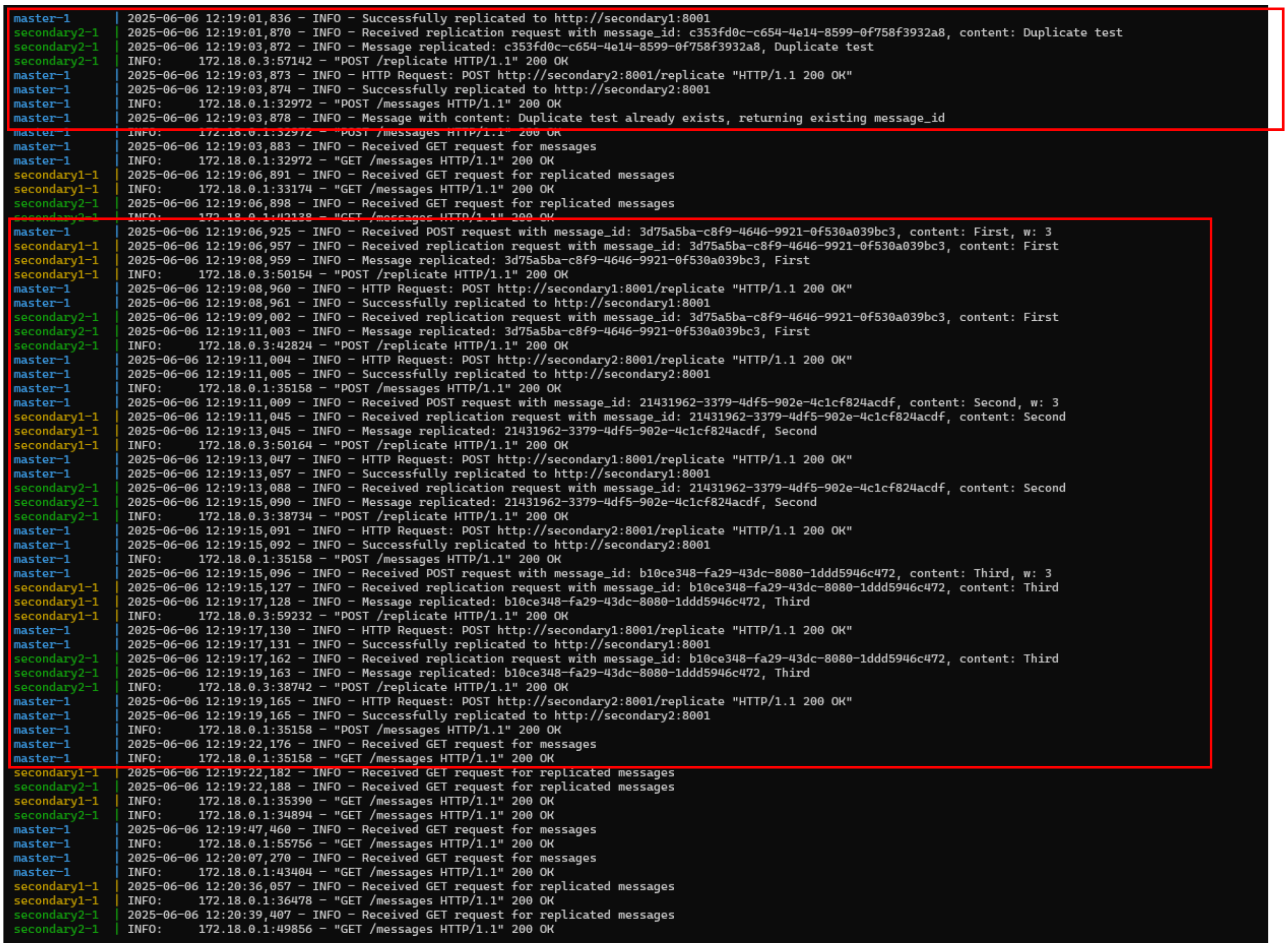
Логи:



```
D:\Documents\dist_systems\project\iteration2>curl http://localhost:8000/messages
{"messages":[{"message_id":"a86b031a-856e-4394-ae7c-8f63cacb9c20","content":"Test w=1"},{"message_id":"7edafdbd-bcce-4842-8628-e34db740b75a","content":"Test
 w=2"},{"message_id":"3c747890-287e-41f8-b8cd-16cdf82dfcb9","content":"Test w=3"},{"message_id":"c353fd0c-c654-4e14-8599-0f758f3932a8","content":"Duplicate
test"},{"message_id":"3d75a5ba-c8f9-4646-9921-0f530a039bc3","content":"First"},{"message_id":"21431962-3379-4df5-902e-4c1cf824acdf","content":"Second"},{"me
ssage_id":"b10ce348-fa29-43dc-8080-1ddd5946c472","content":"Third"}]}
D:\Documents\dist_systems\project\iteration2>curl http://localhost:8001/messages
{"messages":[{"message_id":"a86b031a-856e-4394-ae7c-8f63cacb9c20","content":"Test w=1"},{"message_id":"7edafdbd-bcce-4842-8628-e34db740b75a","content":"Test
 w=2"},{"message_id":"3c747890-287e-41f8-b8cd-16cdf82dfcb9","content":"Test w=3"},{"message_id":"c353fd0c-c654-4e14-8599-0f758f3932a8","content":"Duplicate
test"},{"message_id":"3d75a5ba-c8f9-4646-9921-0f530a039bc3","content":"First"},{"message_id":"21431962-3379-4df5-902e-4c1cf824acdf","content":"Second"},{"me
ssage_id":"b10ce348-fa29-43dc-8080-1ddd5946c472","content":"Third"}]}
D:\Documents\dist_systems\project\iteration2>curl http://localhost:8002/messages
{"messages":[{"message_id":"a86b031a-856e-4394-ae7c-8f63cacb9c20","content":"Test w=1"},{"message_id":"7edafdbd-bcce-4842-8628-e34db740b75a","content":"Test
 w=2"},{"message_id":"3c747890-287e-41f8-b8cd-16cdf82dfcb9","content":"Test w=3"},{"message_id":"c353fd0c-c654-4e14-8599-0f758f3932a8","content":"Duplicate
test"},{"message_id":"3d75a5ba-c8f9-4646-9921-0f530a039bc3","content":"First"},{"message_id":"21431962-3379-4df5-902e-4c1cf824acdf","content":"Second"},{"me
ssage_id":"b10ce348-fa29-43dc-8080-1ddd5946c472","content":"Third"}]}
D:\Documents\dist_systems\project\iteration2>
```



```
master-1       | INFO:     Application startup complete.
master-1       | INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
master-1       | 2025-06-06 12:18:47,451 - INFO - Received POST request with message_id: a86b031a-856e-4394-ae7c-8f63cacb9c20, content: Test w=1, w: 1
master-1       | INFO:     172.18.0.1:38694 - "POST /messages HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:18:47,520 - INFO - Received GET request for messages
master-1       | INFO:     172.18.0.1:38694 - "GET /messages HTTP/1.1" 200 OK
secondary1-1   | 2025-06-06 12:18:47,526 - INFO - Received replication request with message_id: a86b031a-856e-4394-ae7c-8f63cacb9c20, content: Test w=1
secondary2-1   | 2025-06-06 12:18:47,526 - INFO - Received replication request with message_id: a86b031a-856e-4394-ae7c-8f63cacb9c20, content: Test w=1
secondary1-1   | 2025-06-06 12:18:49,527 - INFO - Message replicated: a86b031a-856e-4394-ae7c-8f63cacb9c20, Test w=1
secondary1-1   | INFO:     172.18.0.3:46632 - "POST /replicate HTTP/1.1" 200 OK
secondary2-1   | 2025-06-06 12:18:49,528 - INFO - Message replicated: a86b031a-856e-4394-ae7c-8f63cacb9c20, Test w=1
master-1       | 2025-06-06 12:18:49,529 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
secondary2-1   | INFO:     172.18.0.3:57618 - "POST /replicate HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:18:49,530 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
master-1       | 2025-06-06 12:18:49,531 - INFO - Successfully replicated to http://secondary2:8001
master-1       | 2025-06-06 12:18:49,531 - INFO - Successfully replicated to http://secondary1:8001
secondary1-1   | 2025-06-06 12:18:50,538 - INFO - Received GET request for replicated messages
secondary1-1   | INFO:     172.18.0.1:43726 - "GET /messages HTTP/1.1" 200 OK
secondary2-1   | 2025-06-06 12:18:50,547 - INFO - Received GET request for replicated messages
secondary2-1   | INFO:     172.18.0.1:52866 - "GET /messages HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:18:50,578 - INFO - Received POST request with message_id: 7edafdbd-bcce-4842-8628-e34db740b75a, content: Test w=2, w: 2
secondary1-1   | 2025-06-06 12:18:50,610 - INFO - Received replication request with message_id: 7edafdbd-bcce-4842-8628-e34db740b75a, content: Test w=2
secondary1-1   | 2025-06-06 12:18:52,611 - INFO - Message replicated: 7edafdbd-bcce-4842-8628-e34db740b75a, Test w=2
secondary1-1   | INFO:     172.18.0.3:46642 - "POST /replicate HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:18:52,613 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
master-1       | 2025-06-06 12:18:52,614 - INFO - Successfully replicated to http://secondary1:8001
master-1       | INFO:     172.18.0.1:38704 - "POST /messages HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:18:52,647 - INFO - Received GET request for messages
master-1       | INFO:     172.18.0.1:38704 - "GET /messages HTTP/1.1" 200 OK
secondary2-1   | 2025-06-06 12:18:52,651 - INFO - Received replication request with message_id: 7edafdbd-bcce-4842-8628-e34db740b75a, content: Test w=2
secondary1-1   | 2025-06-06 12:18:52,656 - INFO - Received GET request for replicated messages
secondary2-1   | 2025-06-06 12:18:52,663 - INFO - Received GET request for replicated messages
secondary1-1   | INFO:     172.18.0.1:45586 - "GET /messages HTTP/1.1" 200 OK
secondary2-1   | INFO:     172.18.0.1:57944 - "GET /messages HTTP/1.1" 200 OK
secondary2-1   | 2025-06-06 12:18:54,652 - INFO - Message replicated: 7edafdbd-bcce-4842-8628-e34db740b75a, Test w=2
secondary2-1   | INFO:     172.18.0.3:57120 - "POST /replicate HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:18:54,654 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
master-1       | 2025-06-06 12:18:54,655 - INFO - Successfully replicated to http://secondary2:8001
secondary2-1   | 2025-06-06 12:18:55,668 - INFO - Received GET request for replicated messages
secondary2-1   | INFO:     172.18.0.1:57944 - "GET /messages HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:18:55,692 - INFO - Received POST request with message_id: 3c747890-287e-41f8-b8cd-16cdf82dfcb9, content: Test w=3, w: 3
secondary1-1   | 2025-06-06 12:18:55,723 - INFO - Received replication request with message_id: 3c747890-287e-41f8-b8cd-16cdf82dfcb9, content: Test w=3
secondary1-1   | 2025-06-06 12:18:57,724 - INFO - Message replicated: 3c747890-287e-41f8-b8cd-16cdf82dfcb9, Test w=3
secondary1-1   | INFO:     172.18.0.3:39324 - "POST /replicate HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:18:57,725 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
master-1       | 2025-06-06 12:18:57,726 - INFO - Successfully replicated to http://secondary1:8001
secondary2-1   | 2025-06-06 12:18:57,758 - INFO - Received replication request with message_id: 3c747890-287e-41f8-b8cd-16cdf82dfcb9, content: Test w=3
secondary2-1   | 2025-06-06 12:18:59,759 - INFO - Message replicated: 3c747890-287e-41f8-b8cd-16cdf82dfcb9, Test w=3
secondary2-1   | INFO:     172.18.0.3:57130 - "POST /replicate HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:18:59,760 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
secondary1-1   | 2025-06-06 12:18:59,771 - INFO - Received GET request for replicated messages
master-1       | 2025-06-06 12:18:59,761 - INFO - Successfully replicated to http://secondary2:8001
secondary2-1   | 2025-06-06 12:18:59,777 - INFO - Received GET request for replicated messages
secondary1-1   | INFO:     172.18.0.1:45592 - "GET /messages HTTP/1.1" 200 OK
master-1       | INFO:     172.18.0.1:32968 - "POST /messages HTTP/1.1" 200 OK
secondary2-1   | INFO:     172.18.0.1:57958 - "GET /messages HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:18:59,765 - INFO - Received GET request for messages
master-1       | INFO:     172.18.0.1:32968 - "GET /messages HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:18:59,802 - INFO - Received POST request with message_id: c353fd0c-c654-4e14-8599-0f758f3932a8, content: Duplicate test, w: 3
secondary1-1   | 2025-06-06 12:18:59,834 - INFO - Received replication request with message_id: c353fd0c-c654-4e14-8599-0f758f3932a8, content: Duplicate test
secondary1-1   | 2025-06-06 12:19:01,834 - INFO - Message replicated: c353fd0c-c654-4e14-8599-0f758f3932a8, Duplicate test
secondary1-1   | INFO:     172.18.0.3:39326 - "POST /replicate HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:19:01,835 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
master-1       | 2025-06-06 12:19:01,836 - INFO - Successfully replicated to http://secondary1:8001
secondary2-1   | 2025-06-06 12:19:01,870 - INFO - Received replication request with message_id: c353fd0c-c654-4e14-8599-0f758f3932a8, content: Duplicate test
secondary2-1   | 2025-06-06 12:19:03,872 - INFO - Message replicated: c353fd0c-c654-4e14-8599-0f758f3932a8, Duplicate test
secondary2-1   | INFO:     172.18.0.3:57142 - "POST /replicate HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:19:03,873 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
```



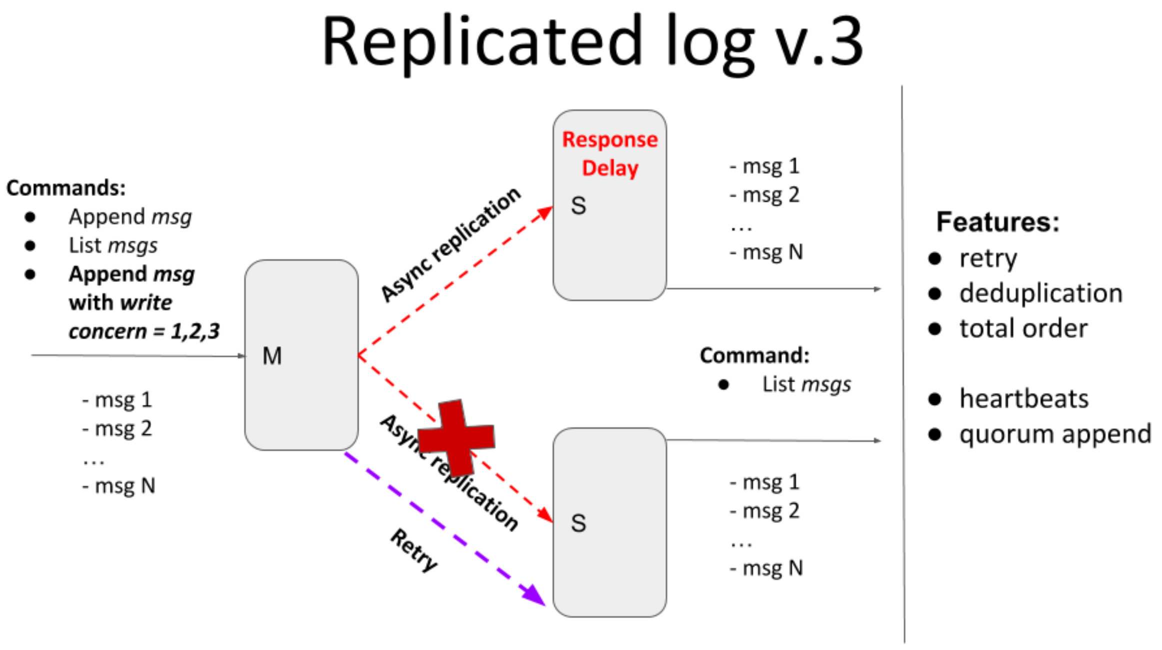
```
master-1       | 2025-06-06 12:19:01,836 - INFO - Successfully replicated to http://secondary1:8001
secondary2-1   | 2025-06-06 12:19:01,870 - INFO - Received replication request with message_id: c353fd0c-c654-4e14-8599-0f758f3932a8, content: Duplicate test
secondary2-1   | 2025-06-06 12:19:03,872 - INFO - Message replicated: c353fd0c-c654-4e14-8599-0f758f3932a8, Duplicate test
secondary2-1   | INFO:     172.18.0.3:57142 - "POST /replicate HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:19:03,873 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
master-1       | 2025-06-06 12:19:03,874 - INFO - Successfully replicated to http://secondary2:8001
master-1       | INFO:     172.18.0.1:32972 - "POST /messages HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:19:03,878 - INFO - Message with content: Duplicate test already exists, returning existing message_id
master-1       | INFO:     172.18.0.1:32972 - "POST /messages HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:19:03,883 - INFO - Received GET request for messages
master-1       | INFO:     172.18.0.1:32972 - "GET /messages HTTP/1.1" 200 OK
secondary1-1   | 2025-06-06 12:19:06,891 - INFO - Received GET request for replicated messages
secondary1-1   | INFO:     172.18.0.1:33174 - "GET /messages HTTP/1.1" 200 OK
secondary2-1   | 2025-06-06 12:19:06,898 - INFO - Received GET request for replicated messages
secondary2-1   | INFO:     172.18.0.1:42138 - "GET /messages HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:19:06,925 - INFO - Received POST request with message_id: 3d75a5ba-c8f9-4646-9921-0f530a039bc3, content: First, w: 3
secondary1-1   | 2025-06-06 12:19:06,957 - INFO - Received replication request with message_id: 3d75a5ba-c8f9-4646-9921-0f530a039bc3, content: First
secondary1-1   | 2025-06-06 12:19:08,959 - INFO - Message replicated: 3d75a5ba-c8f9-4646-9921-0f530a039bc3, First
secondary1-1   | INFO:     172.18.0.3:50154 - "POST /replicate HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:19:08,960 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
master-1       | 2025-06-06 12:19:08,961 - INFO - Successfully replicated to http://secondary1:8001
secondary2-1   | 2025-06-06 12:19:09,002 - INFO - Received replication request with message_id: 3d75a5ba-c8f9-4646-9921-0f530a039bc3, content: First
secondary2-1   | 2025-06-06 12:19:11,003 - INFO - Message replicated: 3d75a5ba-c8f9-4646-9921-0f530a039bc3, First
secondary2-1   | INFO:     172.18.0.3:42824 - "POST /replicate HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:19:11,004 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
master-1       | 2025-06-06 12:19:11,005 - INFO - Successfully replicated to http://secondary2:8001
master-1       | INFO:     172.18.0.1:35158 - "POST /messages HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:19:11,009 - INFO - Received POST request with message_id: 21431962-3379-4df5-902e-4c1cf824acdf, content: Second, w: 3
secondary1-1   | 2025-06-06 12:19:11,045 - INFO - Received replication request with message_id: 21431962-3379-4df5-902e-4c1cf824acdf, content: Second
secondary1-1   | 2025-06-06 12:19:13,045 - INFO - Message replicated: 21431962-3379-4df5-902e-4c1cf824acdf, Second
secondary1-1   | INFO:     172.18.0.3:50164 - "POST /replicate HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:19:13,047 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
master-1       | 2025-06-06 12:19:13,057 - INFO - Successfully replicated to http://secondary1:8001
secondary2-1   | 2025-06-06 12:19:13,088 - INFO - Received replication request with message_id: 21431962-3379-4df5-902e-4c1cf824acdf, content: Second
secondary2-1   | 2025-06-06 12:19:15,090 - INFO - Message replicated: 21431962-3379-4df5-902e-4c1cf824acdf, Second
secondary2-1   | INFO:     172.18.0.3:38734 - "POST /replicate HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:19:15,091 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
master-1       | 2025-06-06 12:19:15,092 - INFO - Successfully replicated to http://secondary2:8001
master-1       | INFO:     172.18.0.1:35158 - "POST /messages HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:19:15,096 - INFO - Received POST request with message_id: b10ce348-fa29-43dc-8080-1ddd5946c472, content: Third, w: 3
secondary1-1   | 2025-06-06 12:19:15,127 - INFO - Received replication request with message_id: b10ce348-fa29-43dc-8080-1ddd5946c472, content: Third
secondary1-1   | 2025-06-06 12:19:17,128 - INFO - Message replicated: b10ce348-fa29-43dc-8080-1ddd5946c472, Third
secondary1-1   | INFO:     172.18.0.3:59232 - "POST /replicate HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:19:17,130 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
master-1       | 2025-06-06 12:19:17,131 - INFO - Successfully replicated to http://secondary1:8001
secondary2-1   | 2025-06-06 12:19:17,162 - INFO - Received replication request with message_id: b10ce348-fa29-43dc-8080-1ddd5946c472, content: Third
secondary2-1   | 2025-06-06 12:19:19,163 - INFO - Message replicated: b10ce348-fa29-43dc-8080-1ddd5946c472, Third
secondary2-1   | INFO:     172.18.0.3:38742 - "POST /replicate HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:19:19,165 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
master-1       | 2025-06-06 12:19:19,165 - INFO - Successfully replicated to http://secondary2:8001
master-1       | INFO:     172.18.0.1:35158 - "POST /messages HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:19:22,176 - INFO - Received GET request for messages
master-1       | INFO:     172.18.0.1:35158 - "GET /messages HTTP/1.1" 200 OK
secondary1-1   | 2025-06-06 12:19:22,182 - INFO - Received GET request for replicated messages
secondary2-1   | 2025-06-06 12:19:22,188 - INFO - Received GET request for replicated messages
secondary1-1   | INFO:     172.18.0.1:35390 - "GET /messages HTTP/1.1" 200 OK
secondary2-1   | INFO:     172.18.0.1:34894 - "GET /messages HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:19:47,460 - INFO - Received GET request for messages
master-1       | INFO:     172.18.0.1:55756 - "GET /messages HTTP/1.1" 200 OK
master-1       | 2025-06-06 12:20:07,270 - INFO - Received GET request for messages
master-1       | INFO:     172.18.0.1:43404 - "GET /messages HTTP/1.1" 200 OK
secondary1-1   | 2025-06-06 12:20:36,057 - INFO - Received GET request for replicated messages
secondary1-1   | INFO:     172.18.0.1:36478 - "GET /messages HTTP/1.1" 200 OK
secondary2-1   | 2025-06-06 12:20:39,407 - INFO - Received GET request for replicated messages
secondary2-1   | INFO:     172.18.0.1:49856 - "GET /messages HTTP/1.1" 200 OK
```

Усі тести виконнао успішно.

- **Master**:
    - Отримує POST-запити, генерує message_id, додає повідомлення.
    - Для w=1: Відповідає одразу, реплікація асинхронна (Secondary додають повідомлення через ~2 секунди).
    - Для w=2: Чекає підтвердження від Secondary1 (~2 секунди).
    - Для w=3: Чекає підтвердження від обох Secondary (~4 секунди).
    - Виконує дедуплікацію (в логах : Message with content: Duplicate test already exists).
- **Secondary**:
    - Отримує реплікаційні запити, додає повідомлення після затримки 2 секунди.
    - Повертає повідомлення через GET у правильному порядку.

Iteration 3.

- **15 points**



The current iteration should provide tunable semi-synchronicity for replication with a *retry* mechanism that should deliver all messages *exactly-once* in total order.

Main features:

- If message delivery fails (due to connection, or internal server error, or secondary is unavailable) the delivery attempts should be repeated - *retry*
  - If one of the secondaries is down and *w=3*, the client should be blocked until the node becomes available. Clients running in parallel shouldn't be blocked by the blocked one.
  - If *w>1* the client should be blocked until the message will be delivered to all secondaries required by the write concern level. Clients running in parallel shouldn't be blocked by the blocked one.
  - All messages that secondaries have missed due to unavailability should be replicated after (re)joining the master
  - Retries can be implemented with an unlimited number of attempts but, possibly, with some "smart" delays logic
  - You can specify a *timeout* for the master in case if there is no response from the secondary
- All messages should be present exactly once in the secondary log - *deduplication*
  - To test deduplication you can generate some random internal server error response from the secondary after the message has been added to the log
- The order of messages should be the same in all nodes - *total order*
  - If secondary has received messages *[msg1, msg2, msg4]*, it shouldn't display the message *'msg4'* until the *'msg3'* will be received
  - To test the total order, you can generate some random internal server error response from the secondaries

Self-check acceptance test:

1. Start M + S1
2. send (Msg1, W=1) - Ok
3. send (Msg2, W=2) - Ok
4. send (Msg3, W=3) - Wait

5. send (Msg4, W=1) - Ok
6. Start S2
7. Check messages on S2 - [Msg1, Msg2, Msg3, Msg4]

Знову реалізовую **master.py:**

```
import logging
import asyncio
import os
import uuid
from fastapi import FastAPI, HTTPException
import httpx
from pydantic import BaseModel
from typing import List, Optional

os.makedirs('/app/logs', exist_ok=True)

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('/app/logs/master.log'),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)

app = FastAPI()

class Message(BaseModel):
    content: str
    w: int

class StoredMessage(BaseModel):
    message_id: str
    content: str
    order: int

class SyncRequest(BaseModel):
    last_message_id: Optional[str]

messages: List[StoredMessage] = []
message_order = 0
SECONDARIES = ["http://secondary1:8001", "http://secondary2:8001"]
RETRY_TIMEOUT = 30
HTTP_TIMEOUT = 10.0
order_lock = asyncio.Lock()

@app.post("/clear")
async def clear_messages():
    global messages, message_order
    async with order_lock:
        messages.clear()
        message_order = 0
    logger.info("Messages cleared")
    logger.info(f"Messages after clear: {messages}, message_order:
{message_order}")
    return {"status": "Messages cleared"}

@app.post("/messages")
async def append_message(message: Message):
    global message_order
    if message.w < 1 or message.w > len(SECONDARIES) + 1:
        logger.error(f"Invalid write concern: w={message.w}")
        raise HTTPException(status_code=400, detail=f"Write concern must be
between 1 and {len(SECONDARIES) + 1}")

    message_id = str(uuid.uuid4())
    if any(m.content == message.content for m in messages):
        logger.info(f"Message with content: {message.content} already exists")
        existing_message = next(m for m in messages if m.content ==
message.content)
        return {"status": "Message already exists", "message_id":
existing_message.message_id, "content": message.content}

    logger.info(f"Received POST request with message_id: {message_id}, content:
{message.content}, w: {message.w}")

    async with order_lock:
        message_order += 1
        messages.append(StoredMessage(message_id=message_id,
content=message.content, order=message_order))

    async def replicate_to_secondary(secondary: str, attempt: int = 1) -> bool:
        backoff = min(2 ** (attempt - 1), 8)
        start_time = asyncio.get_event_loop().time()
        while asyncio.get_event_loop().time() - start_time < RETRY_TIMEOUT:
            async with httpx.AsyncClient(timeout=HTTP_TIMEOUT) as client:
                try:
                    response = await client.post(
                        f"{secondary}/replicate",
                        json={"message_id": message_id, "content":
message.content, "order": message_order}
                    )
                    if response.status_code == 200:
                        logger.info(f"Successfully replicated to {secondary}")
                        return True
                    elif response.status_code == 500:
                        logger.warning(f"Internal server error from {secondary},
retrying after {backoff}s")
                    else:
                        logger.error(f"Failed to replicate to {secondary}:
{response.status_code}")
                except httpx.RequestError as e:
                    logger.error(f"Error replicating to {secondary}: {str(e)}")
            await asyncio.sleep(backoff)
            attempt += 1
        logger.error(f"Retry timeout for {secondary} after {RETRY_TIMEOUT}s")
        return False

    successful_acks = 1
    for i in range(min(message.w - 1, len(SECONDARIES))):
        if await replicate_to_secondary(SECONDARIES[i]):
            successful_acks += 1
        else:
            logger.error(f"Failed to get ACK from {SECONDARIES[i]}")
            raise HTTPException(status_code=500, detail=f"Failed to replicate to
{SECONDARIES[i]}")

    if successful_acks < message.w:
        logger.error(f"Not enough ACKs: got {successful_acks}, required
{message.w}")
        raise HTTPException(status_code=500, detail="Not enough ACKs from
secondaries")

    if message.w - 1 < len(SECONDARIES):
        for secondary in SECONDARIES[message.w - 1:]:
            asyncio.create_task(replicate_to_secondary(secondary))

    return {"status": "Message appended and replicated", "message_id":
message_id, "content": message.content}
```

```
@app.get("/messages")
async def get_messages():
    logger.info("Received GET request for messages")
    return {"messages": [{"message_id": m.message_id, "content": m.content,
"order": m.order} for m in messages]}

@app.post("/sync")
async def sync_messages(sync_request: SyncRequest):
    logger.info(f"Received sync request with last_message_id:
{sync_request.last_message_id}")
    if sync_request.last_message_id is None:
        return {"messages": [{"message_id": m.message_id, "content": m.content,
"order": m.order} for m in messages]}

    try:
        last_message = next(m for m in messages if m.message_id ==
sync_request.last_message_id)
        return {
            "messages": [
                {"message_id": m.message_id, "content": m.content, "order":
m.order}
                for m in messages if m.order > last_message.order
            ]
        }
    except StopIteration:
        logger.error(f"Invalid last_message_id: {sync_request.last_message_id}")
        raise HTTPException(status_code=400, detail="Invalid last_message_id")

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

- Приймає запити на додавання повідомлень (POST /messages) з параметром w, який визначає, скільки вузлів (включаючи Master) повинні підтвердити запис.
- Присвоює кожному повідомленню унікальний message_id та порядок (order).
- Реплікує повідомлення на Secondary вузли асинхронно.
- Підтримує синхронізацію з Secondary через POST /sync.
- Надає список повідомлень через GET /messages.
- Використовує asyncio.Lock для синхронізації message_order, щоб уникнути гонок при асинхронних запитах.
- Логіка реплікації включає повторні спроби (з експоненціальним backoff) у разі помилок.

Secondary.py:

```
import logging
import asyncio
import os
from fastapi import FastAPI, HTTPException
import httpx
import random
from pydantic import BaseModel
from typing import List, Optional

os.makedirs('/app/logs', exist_ok=True)

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('/app/logs/secondary.log'),
        logging.StreamHandler()
    ]
)
logger = logging.getLogger(__name__)

app = FastAPI()

MASTER_URL = "http://master:8000"

class Message(BaseModel):
    message_id: str
    content: str
    order: int

class SyncRequest(BaseModel):
    last_message_id: Optional[str]

replicated_messages: List[Message] = []
max_display_order = 0

async def sync_with_master():
    async with httpx.AsyncClient(timeout=10.0) as client:
        try:
            last_message_id = replicated_messages[-1].message_id if
replicated_messages else None
            response = await client.post(
                f"{MASTER_URL}/sync",
                json={"last_message_id": last_message_id}
            )
            if response.status_code == 200:
                new_messages = response.json()["messages"]
                for msg in sorted(new_messages, key=lambda x: x["order"]):
                    if not any(m.message_id == msg["message_id"] for m in
replicated_messages):
                        replicated_messages.append(Message(**msg))
                        logger.info(f"Synced message: {msg['message_id']},
{msg['content']}, order={msg['order']}")
                global max_display_order
                max_display_order = max(m.order for m in replicated_messages) if
replicated_messages else 0
                logger.info(f"Sync completed,
max_display_order={max_display_order}")
                logger.info(f"Replicated messages after sync: {[(m.message_id,
m.content, m.order) for m in replicated_messages]}")
            else:
                logger.error(f"Sync failed: {response.status_code}")
        except httpx.RequestError as e:
            logger.error(f"Error syncing with master: {str(e)}")

@app.on_event("startup")
async def startup_event():
    logger.info("Starting up, syncing with master")
    await sync_with_master()

@app.post("/clear")
async def clear_messages():
    replicated_messages.clear()
    global max_display_order
    max_display_order = 0
    logger.info("Messages cleared")
    return {"status": "Messages cleared"}

@app.post("/replicate")
async def replicate_message(message: Message):
    logger.info(f"Received replication request with message_id:
{message.message_id}, content: {message.content}, order: {message.order}")
```

```
    if any(m.message_id == message.message_id for m in replicated_messages):
        logger.info(f"Message with message_id: {message.message_id} already
exists, skipping")
        return {"status": "Message already exists"}

    if random.random() < 0.2:
        logger.warning(f"Generating random 500 error for message_id:
{message.message_id}")
        raise HTTPException(status_code=500, detail="Random internal server
error")

    await asyncio.sleep(2)
    replicated_messages.append(message)
    global max_display_order
    max_display_order = max(m.order for m in replicated_messages)
    logger.info(f"Message replicated: {message.message_id}, {message.content},
order={message.order}")
    return {"status": "Message replicated"}

@app.get("/messages")
async def get_messages():
    logger.info("Received GET request for replicated messages")
    logger.info(f"All replicated messages: {[(m.message_id, m.content, m.order)
for m in replicated_messages]}")
    display_messages = []
    expected_order = 1
    for msg in sorted(replicated_messages, key=lambda m: m.order):
        if msg.order == expected_order:
            display_messages.append(msg)
            expected_order += 1
        else:
            logger.warning(f"Order mismatch: expected {expected_order}, got
{msg.order}")
            break
    logger.info(f"Displayed messages: {[(m.message_id, m.content, m.order) for m
in display_messages]}")
    return {"messages": [{"message_id": m.message_id, "content": m.content,
"order": m.order} for m in display_messages]}

if __name__ == "__main__":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8001)
```

- Отримує репліковані повідомлення від Master через POST /replicate.
- Синхронізується з Master при запуску через POST /sync.
- Повертає список реплікованих повідомлень через GET /messages, включаючи лише послідовні повідомлення (без розривів у order).
- Імітує випадкові помилки (20% ймовірність HTTP 500) для тестування стійкості.

Знову пишу тести:

```
import pytest
import httpx
import asyncio
from time import time
import subprocess

MASTER_URL = "http://localhost:8000"
SECONDARY1_URL = "http://localhost:8001"
SECONDARY2_URL = "http://localhost:8002"

@pytest.mark.asyncio
```

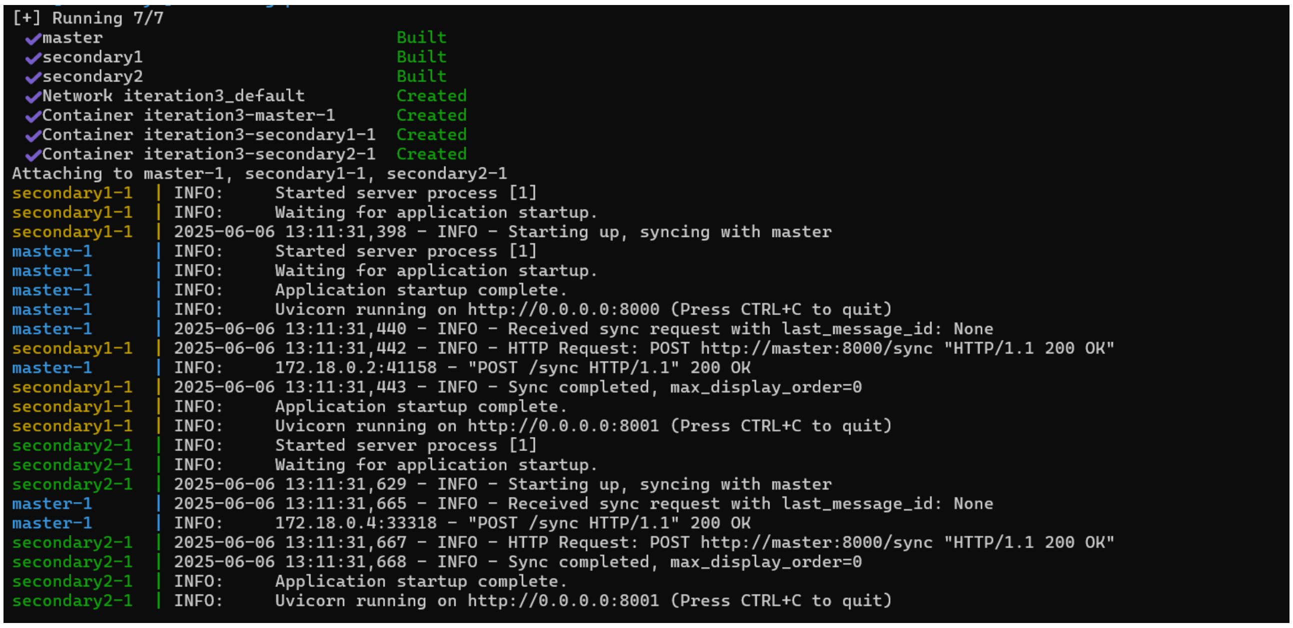
```python
async def test_write_concern_1():
    async with httpx.AsyncClient(timeout=20.0) as client:
        await clear_all(client)
        start_time = time()
        response = await client.post(f"{MASTER_URL}/messages", json={"content":
"Test w=1", "w": 1})
        end_time = time()

        assert response.status_code == 200
        assert response.json()["content"] == "Test w=1"
        assert end_time - start_time < 1.0

        master_response = await client.get(f"{MASTER_URL}/messages")
        assert any(m["content"] == "Test w=1" for m in
master_response.json()["messages"])

        await asyncio.sleep(3)
        for secondary_url in [SECONDARY1_URL, SECONDARY2_URL]:
            secondary_response = await client.get(f"{secondary_url}/messages")
            assert secondary_response.status_code == 200
            assert any(m["content"] == "Test w=1" for m in
secondary_response.json()["messages"])

@pytest.mark.asyncio
async def test_write_concern_2():
    async with httpx.AsyncClient(timeout=20.0) as client:
        await clear_all(client)
        start_time = time()
        response = await client.post(f"{MASTER_URL}/messages", json={"content":
"Test w=2", "w": 2})
        end_time = time()

        assert response.status_code == 200
        assert response.json()["content"] == "Test w=2"
        assert 2.0 <= end_time - start_time < 4.0  # Збільшено межу

        master_response = await client.get(f"{MASTER_URL}/messages")
        assert any(m["content"] == "Test w=2" for m in
master_response.json()["messages"])

        secondary1_response = await client.get(f"{SECONDARY1_URL}/messages")
        assert any(m["content"] == "Test w=2" for m in
secondary1_response.json()["messages"])

        await asyncio.sleep(3)
        secondary2_response = await client.get(f"{SECONDARY2_URL}/messages")
        has_message = any(m["content"] == "Test w=2" for m in
secondary2_response.json()["messages"])
        if not has_message:
            await asyncio.sleep(3)
            secondary2_response = await client.get(f"{SECONDARY2_URL}/messages")
            assert any(m["content"] == "Test w=2" for m in
secondary2_response.json()["messages"])

@pytest.mark.asyncio
async def test_write_concern_3():
    async with httpx.AsyncClient(timeout=20.0) as client:
        await clear_all(client)
        start_time = time()
        response = await client.post(f"{MASTER_URL}/messages", json={"content":
"Test w=3", "w": 3})
        end_time = time()

        assert response.status_code == 200
        assert response.json()["content"] == "Test w=3"
        assert end_time - start_time >= 4.0
        master_response = await client.get(f"{MASTER_URL}/messages")
        assert any(m["content"] == "Test w=3" for m in
master_response.json()["messages"])

        await asyncio.sleep(3)
        for secondary_url in [SECONDARY1_URL, SECONDARY2_URL]:
            secondary_response = await client.get(f"{secondary_url}/messages")
            assert any(m["content"] == "Test w=3" for m in
secondary_response.json()["messages"])

@pytest.mark.asyncio
async def test_deduplication():
    async with httpx.AsyncClient(timeout=20.0) as client:
        await clear_all(client)
        response1 = await client.post(f"{MASTER_URL}/messages", json={"content":
"Duplicate test", "w": 3})
        response2 = await client.post(f"{MASTER_URL}/messages", json={"content":
"Duplicate test", "w": 3})

        master_response = await client.get(f"{MASTER_URL}/messages")
        duplicates = sum(1 for m in master_response.json()["messages"] if
m["content"] == "Duplicate test")
        assert duplicates == 1

        await asyncio.sleep(3)
        for secondary_url in [SECONDARY1_URL, SECONDARY2_URL]:
            secondary_response = await client.get(f"{secondary_url}/messages")
            duplicates = sum(1 for m in secondary_response.json()["messages"] if
m["content"] == "Duplicate test")
            assert duplicates == 1

@pytest.mark.asyncio
async def test_total_ordering():
    async with httpx.AsyncClient(timeout=20.0) as client:
        await clear_all(client)
        messages = ["First", "Second", "Third"]
        for msg in messages:
            await client.post(f"{MASTER_URL}/messages", json={"content": msg,
"w": 3})

        await asyncio.sleep(3)
        master_response = await client.get(f"{MASTER_URL}/messages")
        master_messages = [m["content"] for m in
master_response.json()["messages"]][-3:]
        assert master_messages == messages

        for secondary_url in [SECONDARY1_URL, SECONDARY2_URL]:
            secondary_response = await client.get(f"{secondary_url}/messages")
            secondary_messages = [m["content"] for m in
secondary_response.json()["messages"]][-3:]
            assert secondary_messages == messages

@pytest.mark.asyncio
async def test_acceptance():
    async with httpx.AsyncClient(timeout=20.0) as client:
        await clear_all(client)

        subprocess.run(["docker-compose", "stop", "secondary2"], check=True)

        response1 = await client.post(f"{MASTER_URL}/messages", json={"content":
"Msg1", "w": 1})
        assert response1.status_code == 200
        assert response1.json()["content"] == "Msg1"

        response2 = await client.post(f"{MASTER_URL}/messages", json={"content":
```

```
"Msg2", "w": 2})
        assert response2.status_code == 200
        assert response2.json()["content"] == "Msg2"

        async def send_msg3():
            start_time = time()
            response = await client.post(f"{MASTER_URL}/messages",
json={"content": "Msg3", "w": 3})
            assert response.status_code == 200
            assert response.json()["content"] == "Msg3"
            assert time() - start_time > 2.0
            return response.json()["message_id"]
        msg3_task = asyncio.create_task(send_msg3())

        await asyncio.sleep(0.1)
        response4 = await client.post(f"{MASTER_URL}/messages", json={"content":
"Msg4", "w": 1})
        assert response4.status_code == 200
        assert response4.json()["content"] == "Msg4"

        subprocess.run(["docker-compose", "start", "secondary2"], check=True)

        msg3_message_id = await msg3_task

        await asyncio.sleep(10)

        secondary2_response = await client.get(f"{SECONDARY2_URL}/messages")
        secondary2_messages = [m["content"] for m in
secondary2_response.json()["messages"]]
        assert secondary2_messages == ["Msg1", "Msg2", "Msg3", "Msg4"]

        secondary2_message_ids = [m["message_id"] for m in
secondary2_response.json()["messages"]]
        assert len(secondary2_message_ids) == len(set(secondary2_message_ids))
# Унікальні message_id
        assert msg3_message_id in secondary2_message_ids

async def clear_all(client):
    try:
        await client.post(f"{MASTER_URL}/clear")
        await client.post(f"{SECONDARY1_URL}/clear")
        await client.post(f"{SECONDARY2_URL}/clear")
    except httpx.RequestError:
        pass
```

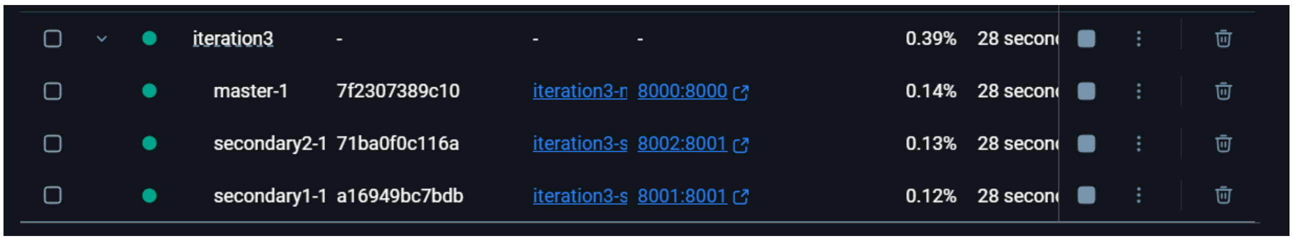
1. **test_write_concern_1** - перевіряє, що повідомлення з w=1 записується на Master і реплікується на обидва Secondary.
2. **test_write_concern_2** - перевіряє, що повідомлення з w=2 записується на Master і один Secondary.
3. **test_write_concern_3** - перевіряє, що повідомлення з w=3 записується на Master і обидва Secondary.
4. **test_deduplication** - перевіряє, що повторне відправлення повідомлення з однаковим вмістом не створює дублікатів.
5. **test_total_ordering** - перевіряє, що повідомлення зберігаються в однаковому порядку на всіх вузлах.
6. **test_acceptance** - перевіряє стійкість системи до відмови одного Secondary з подальшим відновленням.
    - **Послідовність виконання:**
        - Очищення стану.
        - Зупинка secondary2 через docker-compose stop.

- Відправка Msg1 (w=1), Msg2 (w=2), Msg3 (w=3, асинхронно), і Msg4 (w=1) з затримкою 0.1 с між Msg3 і Msg4.
- Запуск secondary2.
- Очікування синхронізації (10 с).
- Перевірка, що secondary2 повертає ['Msg1', 'Msg2', 'Msg3', 'Msg4'] і що message_id для Msg3 є в списку.

Запуск:

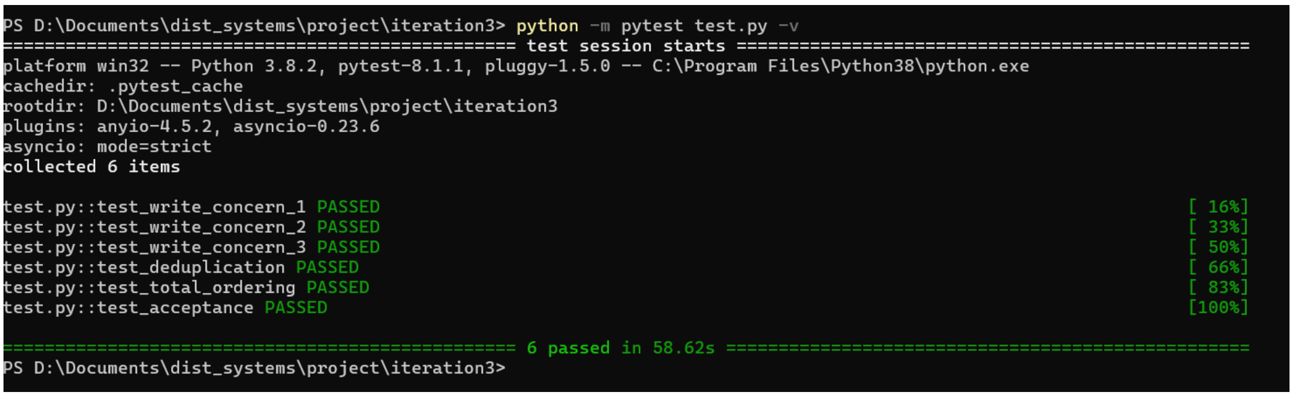


```
[+] Running 7/7
 ✔master                           Built
 ✔secondary1                       Built
 ✔secondary2                       Built
 ✔Network iteration3_default       Created
 ✔Container iteration3-master-1    Created
 ✔Container iteration3-secondary1-1 Created
 ✔Container iteration3-secondary2-1 Created
Attaching to master-1, secondary1-1, secondary2-1
secondary1-1  | INFO:     Started server process [1]
secondary1-1  | INFO:     Waiting for application startup.
secondary1-1  | 2025-06-06 13:11:31,398 - INFO - Starting up, syncing with master
master-1      | INFO:     Started server process [1]
master-1      | INFO:     Waiting for application startup.
master-1      | INFO:     Application startup complete.
master-1      | INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
master-1      | 2025-06-06 13:11:31,440 - INFO - Received sync request with last_message_id: None
secondary1-1  | 2025-06-06 13:11:31,442 - INFO - HTTP Request: POST http://master:8000/sync "HTTP/1.1 200 OK"
master-1      | INFO:     172.18.0.2:41158 - "POST /sync HTTP/1.1" 200 OK
secondary1-1  | 2025-06-06 13:11:31,443 - INFO - Sync completed, max_display_order=0
secondary1-1  | INFO:     Application startup complete.
secondary1-1  | INFO:     Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
secondary2-1  | INFO:     Started server process [1]
secondary2-1  | INFO:     Waiting for application startup.
secondary2-1  | 2025-06-06 13:11:31,629 - INFO - Starting up, syncing with master
master-1      | 2025-06-06 13:11:31,665 - INFO - Received sync request with last_message_id: None
master-1      | INFO:     172.18.0.4:33318 - "POST /sync HTTP/1.1" 200 OK
secondary2-1  | 2025-06-06 13:11:31,667 - INFO - HTTP Request: POST http://master:8000/sync "HTTP/1.1 200 OK"
secondary2-1  | 2025-06-06 13:11:31,668 - INFO - Sync completed, max_display_order=0
secondary2-1  | INFO:     Application startup complete.
secondary2-1  | INFO:     Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
```



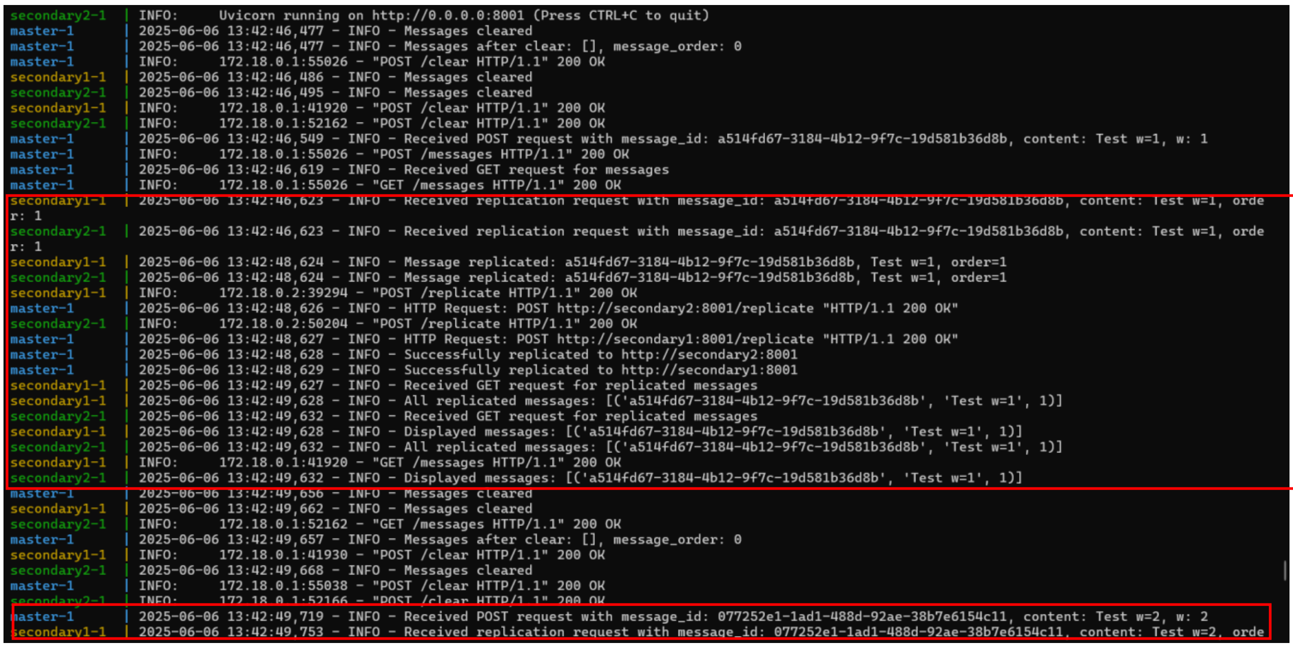
	Name	Container ID	Image	Port(s)	CPU	Last started
●	iteration3	-	-	-	0.39%	28 secon
●	master-1	7f2307389c10	iteration3-n	8000:8000	0.14%	28 secon
●	secondary2-1	71ba0f0c116a	iteration3-s	8002:8001	0.13%	28 secon
●	secondary1-1	a16949bc7bdb	iteration3-s	8001:8001	0.12%	28 secon

Виконання тестів:

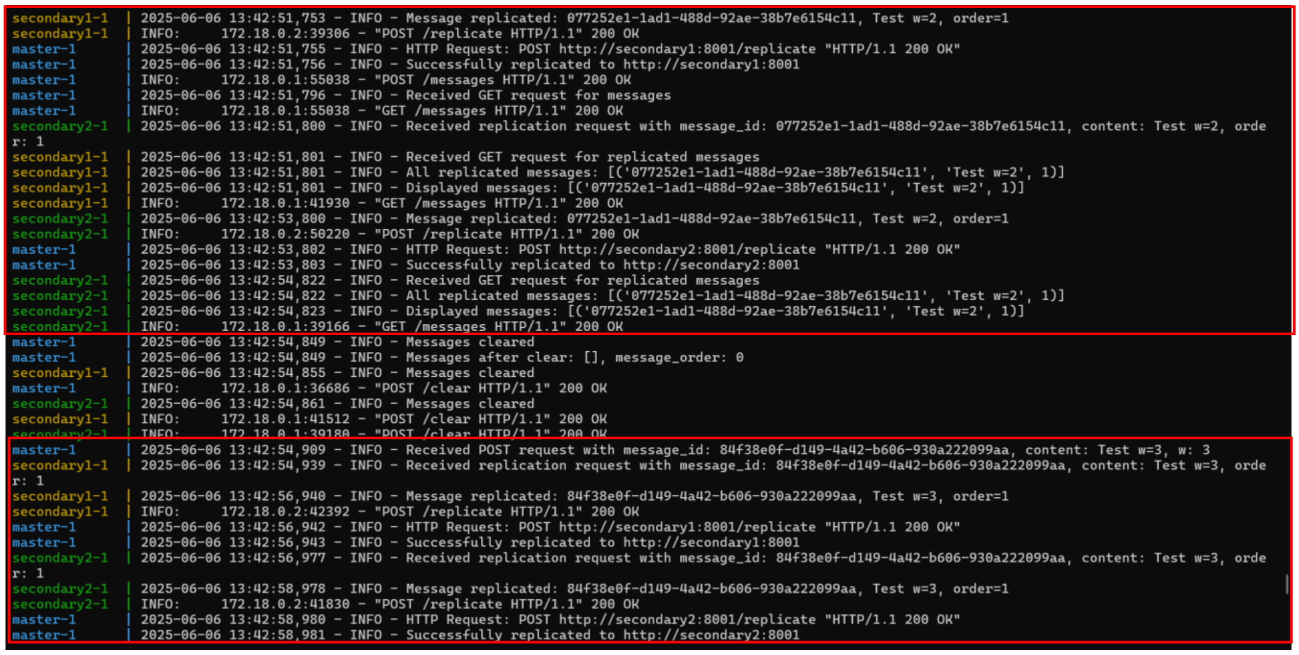


```
PS D:\Documents\dist_systems\project\iteration3> python -m pytest test.py -v
================================================= test session starts =================================================
platform win32 -- Python 3.8.2, pytest-8.1.1, pluggy-1.5.0 -- C:\Program Files\Python38\python.exe
cachedir: .pytest_cache
rootdir: D:\Documents\dist_systems\project\iteration3
plugins: anyio-4.5.2, asyncio-0.23.6
asyncio: mode=strict
collected 6 items

test.py::test_write_concern_1 PASSED                                                                            [ 16%]
test.py::test_write_concern_2 PASSED                                                                            [ 33%]
test.py::test_write_concern_3 PASSED                                                                            [ 50%]
test.py::test_deduplication PASSED                                                                              [ 66%]
test.py::test_total_ordering PASSED                                                                             [ 83%]
test.py::test_acceptance PASSED                                                                                 [100%]

================================================= 6 passed in 58.62s ==================================================
PS D:\Documents\dist_systems\project\iteration3>
```

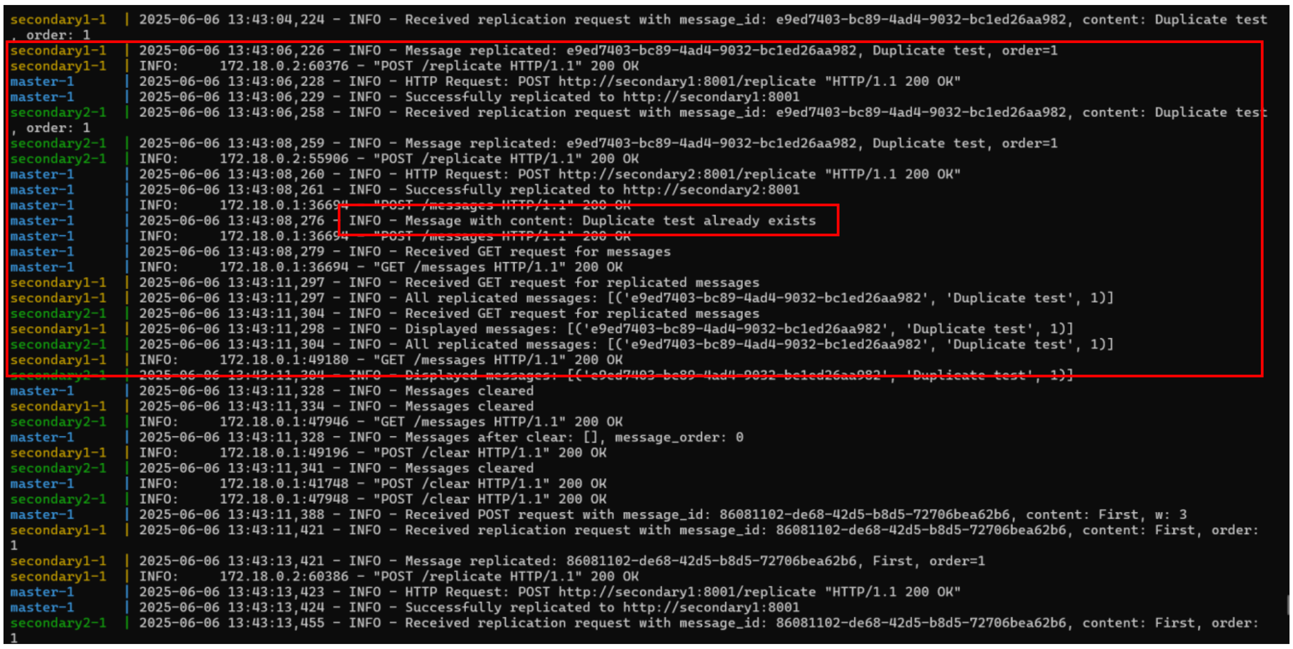
Логи:



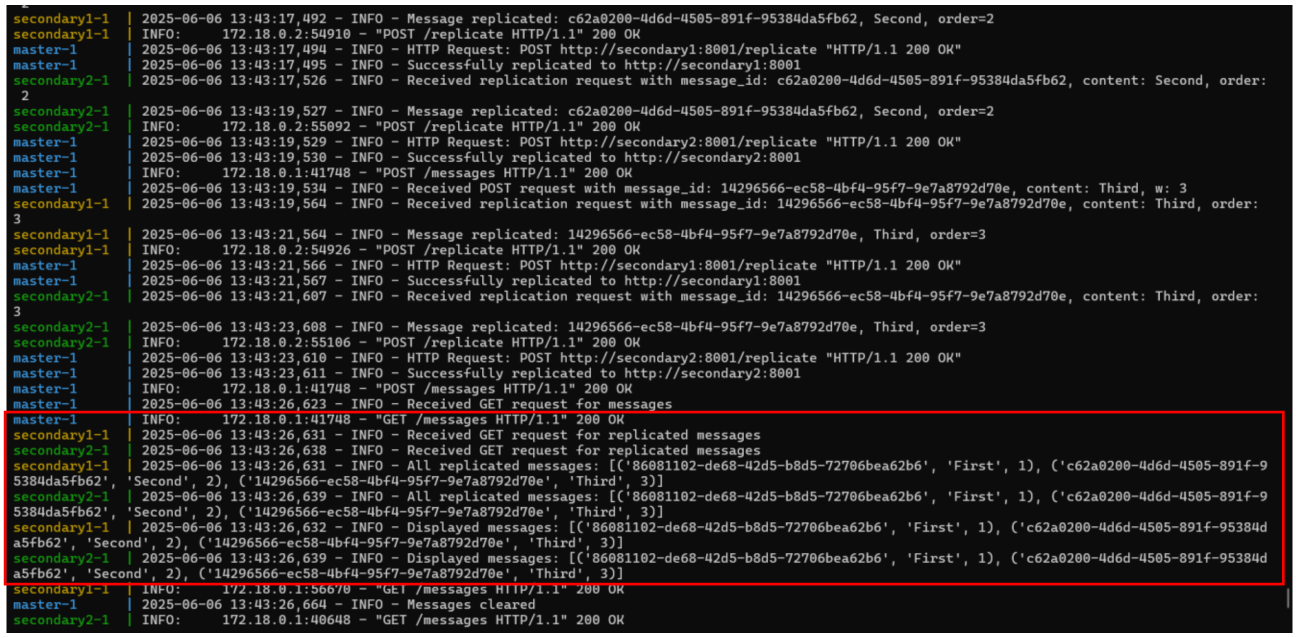
```
secondary2-1  | INFO:     Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
master-1      | 2025-06-06 13:42:46,477 - INFO - Messages cleared
master-1      | 2025-06-06 13:42:46,477 - INFO - Messages after clear: [], message_order: 0
master-1      | INFO:     172.18.0.1:55026 - "POST /clear HTTP/1.1" 200 OK
secondary1-1  | 2025-06-06 13:42:46,486 - INFO - Messages cleared
secondary2-1  | 2025-06-06 13:42:46,495 - INFO - Messages cleared
secondary1-1  | INFO:     172.18.0.1:41920 - "POST /clear HTTP/1.1" 200 OK
secondary2-1  | INFO:     172.18.0.1:52162 - "POST /clear HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:42:46,549 - INFO - Received POST request with message_id: a514fd67-3184-4b12-9f7c-19d581b36d8b, content: Test w=1, w: 1
master-1      | INFO:     172.18.0.1:55026 - "POST /messages HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:42:46,619 - INFO - Received GET request for messages
master-1      | INFO:     172.18.0.1:55026 - "GET /messages HTTP/1.1" 200 OK
secondary1-1  | 2025-06-06 13:42:46,623 - INFO - Received replication request with message_id: a514fd67-3184-4b12-9f7c-19d581b36d8b, content: Test w=1, orde
r: 1
secondary2-1  | 2025-06-06 13:42:46,623 - INFO - Received replication request with message_id: a514fd67-3184-4b12-9f7c-19d581b36d8b, content: Test w=1, orde
r: 1
secondary1-1  | 2025-06-06 13:42:48,624 - INFO - Message replicated: a514fd67-3184-4b12-9f7c-19d581b36d8b, Test w=1, order=1
secondary2-1  | 2025-06-06 13:42:48,624 - INFO - Message replicated: a514fd67-3184-4b12-9f7c-19d581b36d8b, Test w=1, order=1
secondary1-1  | INFO:     172.18.0.2:39294 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:42:48,626 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
secondary2-1  | INFO:     172.18.0.2:50204 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:42:48,627 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
master-1      | 2025-06-06 13:42:48,628 - INFO - Successfully replicated to http://secondary2:8001
master-1      | 2025-06-06 13:42:48,629 - INFO - Successfully replicated to http://secondary1:8001
secondary1-1  | 2025-06-06 13:42:49,627 - INFO - Received GET request for replicated messages
secondary1-1  | 2025-06-06 13:42:49,628 - INFO - All replicated messages: [('a514fd67-3184-4b12-9f7c-19d581b36d8b', 'Test w=1', 1)]
secondary2-1  | 2025-06-06 13:42:49,632 - INFO - Received GET request for replicated messages
secondary1-1  | 2025-06-06 13:42:49,628 - INFO - Displayed messages: [('a514fd67-3184-4b12-9f7c-19d581b36d8b', 'Test w=1', 1)]
secondary2-1  | 2025-06-06 13:42:49,632 - INFO - All replicated messages: [('a514fd67-3184-4b12-9f7c-19d581b36d8b', 'Test w=1', 1)]
secondary1-1  | INFO:     172.18.0.1:41920 - "GET /messages HTTP/1.1" 200 OK
secondary2-1  | 2025-06-06 13:42:49,632 - INFO - Displayed messages: [('a514fd67-3184-4b12-9f7c-19d581b36d8b', 'Test w=1', 1)]
master-1      | 2025-06-06 13:42:49,656 - INFO - Messages cleared
secondary1-1  | 2025-06-06 13:42:49,662 - INFO - Messages cleared
secondary2-1  | INFO:     172.18.0.1:52162 - "GET /messages HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:42:49,657 - INFO - Messages after clear: [], message_order: 0
secondary1-1  | INFO:     172.18.0.1:41930 - "POST /clear HTTP/1.1" 200 OK
secondary2-1  | 2025-06-06 13:42:49,668 - INFO - Messages cleared
master-1      | INFO:     172.18.0.1:55038 - "POST /clear HTTP/1.1" 200 OK
secondary2-1  | INFO:     172.18.0.1:52166 - "POST /clear HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:42:49,719 - INFO - Received POST request with message_id: 077252e1-1ad1-488d-92ae-38b7e6154c11, content: Test w=2, w: 2
secondary1-1  | 2025-06-06 13:42:49,753 - INFO - Received replication request with message_id: 077252e1-1ad1-488d-92ae-38b7e6154c11, content: Test w=2, orde
```



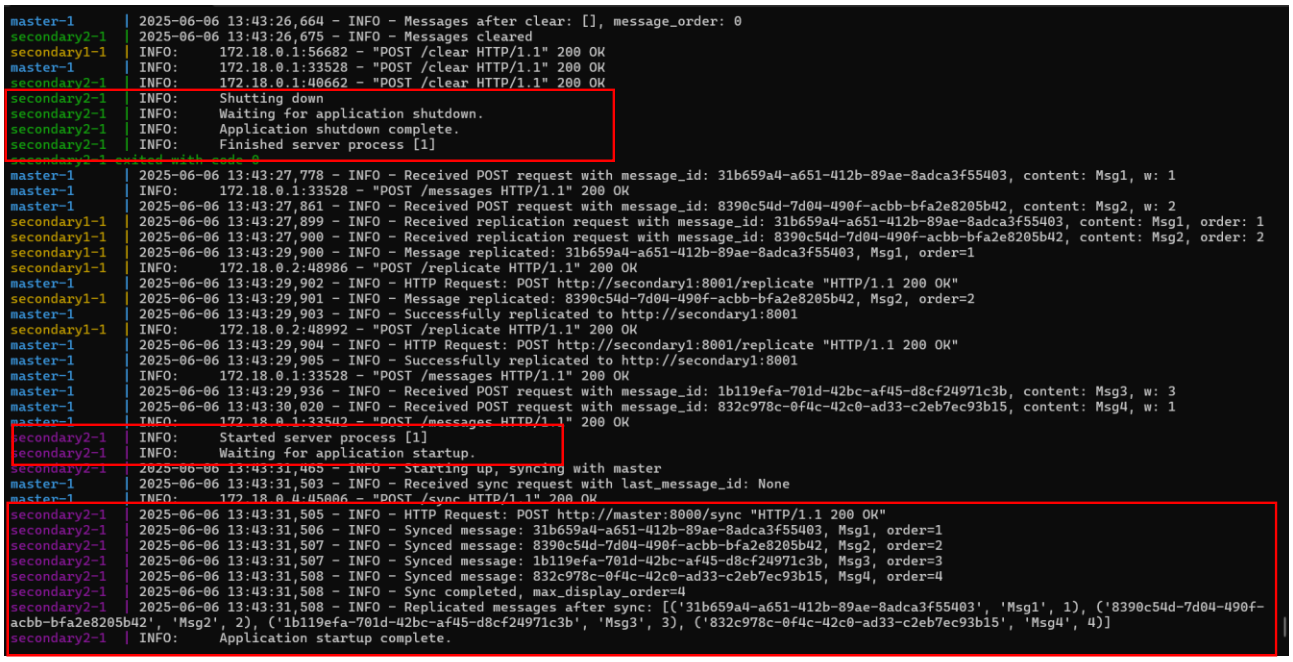
```
secondary1-1  | 2025-06-06 13:42:51,753 - INFO - Message replicated: 077252e1-1ad1-488d-92ae-38b7e6154c11, Test w=2, order=1
secondary1-1  | INFO:     172.18.0.2:39306 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:42:51,755 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
master-1      | 2025-06-06 13:42:51,756 - INFO - Successfully replicated to http://secondary1:8001
master-1      | INFO:     172.18.0.1:55038 - "POST /messages HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:42:51,796 - INFO - Received GET request for messages
master-1      | INFO:     172.18.0.1:55038 - "GET /messages HTTP/1.1" 200 OK
secondary2-1  | 2025-06-06 13:42:51,800 - INFO - Received replication request with message_id: 077252e1-1ad1-488d-92ae-38b7e6154c11, content: Test w=2, orde
r: 1
secondary1-1  | 2025-06-06 13:42:51,801 - INFO - Received GET request for replicated messages
secondary1-1  | 2025-06-06 13:42:51,801 - INFO - All replicated messages: [('077252e1-1ad1-488d-92ae-38b7e6154c11', 'Test w=2', 1)]
secondary1-1  | 2025-06-06 13:42:51,801 - INFO - Displayed messages: [('077252e1-1ad1-488d-92ae-38b7e6154c11', 'Test w=2', 1)]
secondary1-1  | INFO:     172.18.0.1:41930 - "GET /messages HTTP/1.1" 200 OK
secondary2-1  | 2025-06-06 13:42:53,800 - INFO - Message replicated: 077252e1-1ad1-488d-92ae-38b7e6154c11, Test w=2, order=1
secondary2-1  | INFO:     172.18.0.2:50220 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:42:53,802 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
master-1      | 2025-06-06 13:42:53,803 - INFO - Successfully replicated to http://secondary2:8001
secondary2-1  | 2025-06-06 13:42:54,822 - INFO - Received GET request for replicated messages
secondary2-1  | 2025-06-06 13:42:54,822 - INFO - All replicated messages: [('077252e1-1ad1-488d-92ae-38b7e6154c11', 'Test w=2', 1)]
secondary2-1  | 2025-06-06 13:42:54,823 - INFO - Displayed messages: [('077252e1-1ad1-488d-92ae-38b7e6154c11', 'Test w=2', 1)]
secondary2-1  | INFO:     172.18.0.1:39166 - "GET /messages HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:42:54,849 - INFO - Messages cleared
master-1      | 2025-06-06 13:42:54,849 - INFO - Messages after clear: [], message_order: 0
secondary1-1  | 2025-06-06 13:42:54,855 - INFO - Messages cleared
master-1      | INFO:     172.18.0.1:36686 - "POST /clear HTTP/1.1" 200 OK
secondary2-1  | 2025-06-06 13:42:54,861 - INFO - Messages cleared
secondary1-1  | INFO:     172.18.0.1:41512 - "POST /clear HTTP/1.1" 200 OK
secondary2-1  | INFO:     172.18.0.1:39180 - "POST /clear HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:42:54,909 - INFO - Received POST request with message_id: 84f38e0f-d149-4a42-b606-930a222099aa, content: Test w=3, w: 3
secondary1-1  | 2025-06-06 13:42:54,939 - INFO - Received replication request with message_id: 84f38e0f-d149-4a42-b606-930a222099aa, content: Test w=3, orde
r: 1
secondary1-1  | 2025-06-06 13:42:56,940 - INFO - Message replicated: 84f38e0f-d149-4a42-b606-930a222099aa, Test w=3, order=1
secondary1-1  | INFO:     172.18.0.2:42392 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:42:56,942 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
master-1      | 2025-06-06 13:42:56,943 - INFO - Successfully replicated to http://secondary1:8001
secondary2-1  | 2025-06-06 13:42:56,977 - INFO - Received replication request with message_id: 84f38e0f-d149-4a42-b606-930a222099aa, content: Test w=3, orde
r: 1
secondary2-1  | 2025-06-06 13:42:58,978 - INFO - Message replicated: 84f38e0f-d149-4a42-b606-930a222099aa, Test w=3, order=1
secondary2-1  | INFO:     172.18.0.2:41830 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:42:58,980 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
master-1      | 2025-06-06 13:42:58,981 - INFO - Successfully replicated to http://secondary2:8001
```



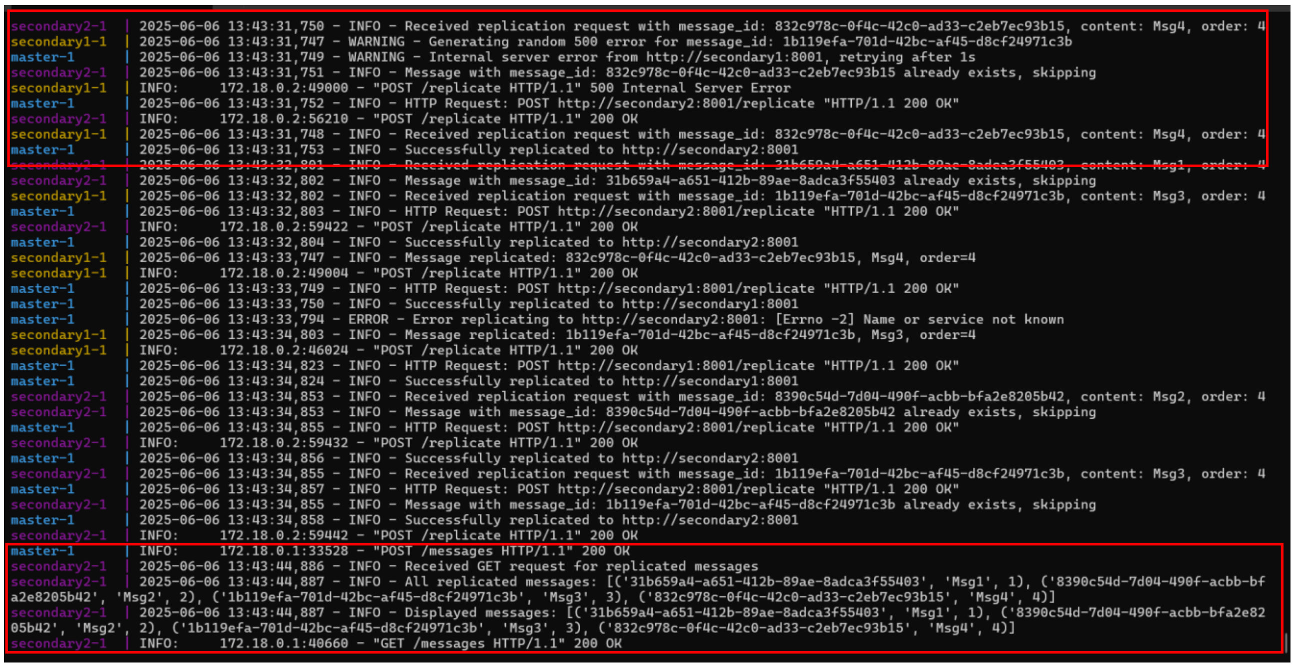
```
secondary1-1  | 2025-06-06 13:43:04,224 - INFO - Received replication request with message_id: e9ed7403-bc89-4ad4-9032-bc1ed26aa982, content: Duplicate test
, order: 1
secondary1-1  | 2025-06-06 13:43:06,226 - INFO - Message replicated: e9ed7403-bc89-4ad4-9032-bc1ed26aa982, Duplicate test, order=1
secondary1-1  | INFO:     172.18.0.2:60376 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:06,228 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
master-1      | 2025-06-06 13:43:06,229 - INFO - Successfully replicated to http://secondary1:8001
secondary2-1  | 2025-06-06 13:43:06,258 - INFO - Received replication request with message_id: e9ed7403-bc89-4ad4-9032-bc1ed26aa982, content: Duplicate test
, order: 1
secondary2-1  | 2025-06-06 13:43:08,259 - INFO - Message replicated: e9ed7403-bc89-4ad4-9032-bc1ed26aa982, Duplicate test, order=1
secondary2-1  | INFO:     172.18.0.2:55906 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:08,260 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
master-1      | 2025-06-06 13:43:08,261 - INFO - Successfully replicated to http://secondary2:8001
master-1      | INFO:     172.18.0.1:36694 - "POST /messages HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:08,276 - INFO - Message with content: Duplicate test already exists
master-1      | INFO:     172.18.0.1:36694 - "POST /messages HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:08,279 - INFO - Received GET request for messages
master-1      | INFO:     172.18.0.1:36694 - "GET /messages HTTP/1.1" 200 OK
secondary1-1  | 2025-06-06 13:43:11,297 - INFO - Received GET request for replicated messages
secondary1-1  | 2025-06-06 13:43:11,297 - INFO - All replicated messages: [('e9ed7403-bc89-4ad4-9032-bc1ed26aa982', 'Duplicate test', 1)]
secondary2-1  | 2025-06-06 13:43:11,304 - INFO - Received GET request for replicated messages
secondary1-1  | 2025-06-06 13:43:11,298 - INFO - Displayed messages: [('e9ed7403-bc89-4ad4-9032-bc1ed26aa982', 'Duplicate test', 1)]
secondary2-1  | 2025-06-06 13:43:11,304 - INFO - All replicated messages: [('e9ed7403-bc89-4ad4-9032-bc1ed26aa982', 'Duplicate test', 1)]
secondary1-1  | INFO:     172.18.0.1:49180 - "GET /messages HTTP/1.1" 200 OK
secondary2-1  | 2025-06-06 13:43:11,304 - INFO - Displayed messages: [('e9ed7403-bc89-4ad4-9032-bc1ed26aa982', 'Duplicate test', 1)]
master-1      | 2025-06-06 13:43:11,328 - INFO - Messages cleared
secondary1-1  | 2025-06-06 13:43:11,334 - INFO - Messages cleared
secondary2-1  | INFO:     172.18.0.1:47946 - "GET /messages HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:11,328 - INFO - Messages after clear: [], message_order: 0
secondary1-1  | INFO:     172.18.0.1:49196 - "POST /clear HTTP/1.1" 200 OK
secondary2-1  | 2025-06-06 13:43:11,341 - INFO - Messages cleared
master-1      | INFO:     172.18.0.1:41748 - "POST /clear HTTP/1.1" 200 OK
secondary2-1  | INFO:     172.18.0.1:47948 - "POST /clear HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:11,388 - INFO - Received POST request with message_id: 86081102-de68-42d5-b8d5-72706bea62b6, content: First, w: 3
secondary1-1  | 2025-06-06 13:43:11,421 - INFO - Received replication request with message_id: 86081102-de68-42d5-b8d5-72706bea62b6, content: First, order:
1
secondary1-1  | 2025-06-06 13:43:13,421 - INFO - Message replicated: 86081102-de68-42d5-b8d5-72706bea62b6, First, order=1
secondary1-1  | INFO:     172.18.0.2:60386 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:13,423 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
master-1      | 2025-06-06 13:43:13,424 - INFO - Successfully replicated to http://secondary1:8001
secondary2-1  | 2025-06-06 13:43:13,455 - INFO - Received replication request with message_id: 86081102-de68-42d5-b8d5-72706bea62b6, content: First, order:
1
```



```
secondary1-1  | 2025-06-06 13:43:17,492 - INFO - Message replicated: c62a0200-4d6d-4505-891f-95384da5fb62, Second, order=2
secondary1-1  | INFO:     172.18.0.2:54910 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:17,494 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
master-1      | 2025-06-06 13:43:17,495 - INFO - Successfully replicated to http://secondary1:8001
secondary2-1  | 2025-06-06 13:43:17,526 - INFO - Received replication request with message_id: c62a0200-4d6d-4505-891f-95384da5fb62, content: Second, order:
 2
secondary2-1  | 2025-06-06 13:43:19,527 - INFO - Message replicated: c62a0200-4d6d-4505-891f-95384da5fb62, Second, order=2
secondary2-1  | INFO:     172.18.0.2:55092 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:19,529 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
master-1      | 2025-06-06 13:43:19,530 - INFO - Successfully replicated to http://secondary2:8001
master-1      | INFO:     172.18.0.1:41748 - "POST /messages HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:19,534 - INFO - Received POST request with message_id: 14296566-ec58-4bf4-95f7-9e7a8792d70e, content: Third, w: 3
secondary1-1  | 2025-06-06 13:43:19,564 - INFO - Received replication request with message_id: 14296566-ec58-4bf4-95f7-9e7a8792d70e, content: Third, order:
3
secondary1-1  | 2025-06-06 13:43:21,564 - INFO - Message replicated: 14296566-ec58-4bf4-95f7-9e7a8792d70e, Third, order=3
secondary1-1  | INFO:     172.18.0.2:54926 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:21,566 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
master-1      | 2025-06-06 13:43:21,567 - INFO - Successfully replicated to http://secondary1:8001
secondary2-1  | 2025-06-06 13:43:21,607 - INFO - Received replication request with message_id: 14296566-ec58-4bf4-95f7-9e7a8792d70e, content: Third, order:
3
secondary2-1  | 2025-06-06 13:43:23,608 - INFO - Message replicated: 14296566-ec58-4bf4-95f7-9e7a8792d70e, Third, order=3
secondary2-1  | INFO:     172.18.0.2:55106 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:23,610 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
master-1      | 2025-06-06 13:43:23,611 - INFO - Successfully replicated to http://secondary2:8001
master-1      | INFO:     172.18.0.1:41748 - "POST /messages HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:26,623 - INFO - Received GET request for messages
master-1      | INFO:     172.18.0.1:41748 - "GET /messages HTTP/1.1" 200 OK
secondary1-1  | 2025-06-06 13:43:26,631 - INFO - Received GET request for replicated messages
secondary2-1  | 2025-06-06 13:43:26,638 - INFO - Received GET request for replicated messages
secondary1-1  | 2025-06-06 13:43:26,631 - INFO - All replicated messages: [('86081102-de68-42d5-b8d5-72706bea62b6', 'First', 1), ('c62a0200-4d6d-4505-891f-9
5384da5fb62', 'Second', 2), ('14296566-ec58-4bf4-95f7-9e7a8792d70e', 'Third', 3)]
secondary2-1  | 2025-06-06 13:43:26,639 - INFO - All replicated messages: [('86081102-de68-42d5-b8d5-72706bea62b6', 'First', 1), ('c62a0200-4d6d-4505-891f-9
5384da5fb62', 'Second', 2), ('14296566-ec58-4bf4-95f7-9e7a8792d70e', 'Third', 3)]
secondary1-1  | 2025-06-06 13:43:26,632 - INFO - Displayed messages: [('86081102-de68-42d5-b8d5-72706bea62b6', 'First', 1), ('c62a0200-4d6d-4505-891f-95384d
a5fb62', 'Second', 2), ('14296566-ec58-4bf4-95f7-9e7a8792d70e', 'Third', 3)]
secondary2-1  | 2025-06-06 13:43:26,639 - INFO - Displayed messages: [('86081102-de68-42d5-b8d5-72706bea62b6', 'First', 1), ('c62a0200-4d6d-4505-891f-95384d
a5fb62', 'Second', 2), ('14296566-ec58-4bf4-95f7-9e7a8792d70e', 'Third', 3)]
secondary1-1  | INFO:     172.18.0.1:56670 - "GET /messages HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:26,664 - INFO - Messages cleared
secondary2-1  | INFO:     172.18.0.1:40648 - "GET /messages HTTP/1.1" 200 OK
```



```
master-1      | 2025-06-06 13:43:26,664 - INFO - Messages after clear: [], message_order: 0
secondary2-1  | 2025-06-06 13:43:26,675 - INFO - Messages cleared
secondary1-1  | INFO:     172.18.0.1:56682 - "POST /clear HTTP/1.1" 200 OK
master-1      | INFO:     172.18.0.1:33528 - "POST /clear HTTP/1.1" 200 OK
secondary2-1  | INFO:     172.18.0.1:40662 - "POST /clear HTTP/1.1" 200 OK
secondary2-1  | INFO:     Shutting down
secondary2-1  | INFO:     Waiting for application shutdown.
secondary2-1  | INFO:     Application shutdown complete.
secondary2-1  | INFO:     Finished server process [1]
secondary2-1 exited with code 0
master-1      | 2025-06-06 13:43:27,778 - INFO - Received POST request with message_id: 31b659a4-a651-412b-89ae-8adca3f55403, content: Msg1, w: 1
master-1      | INFO:     172.18.0.1:33528 - "POST /messages HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:27,861 - INFO - Received POST request with message_id: 8390c54d-7d04-490f-acbb-bfa2e8205b42, content: Msg2, w: 2
secondary1-1  | 2025-06-06 13:43:27,899 - INFO - Received replication request with message_id: 31b659a4-a651-412b-89ae-8adca3f55403, content: Msg1, order: 1
secondary1-1  | 2025-06-06 13:43:27,900 - INFO - Received replication request with message_id: 8390c54d-7d04-490f-acbb-bfa2e8205b42, content: Msg2, order: 2
secondary1-1  | 2025-06-06 13:43:29,900 - INFO - Message replicated: 31b659a4-a651-412b-89ae-8adca3f55403, Msg1, order=1
secondary1-1  | INFO:     172.18.0.2:48986 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:29,902 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
secondary1-1  | 2025-06-06 13:43:29,901 - INFO - Message replicated: 8390c54d-7d04-490f-acbb-bfa2e8205b42, Msg2, order=2
master-1      | 2025-06-06 13:43:29,903 - INFO - Successfully replicated to http://secondary1:8001
secondary1-1  | INFO:     172.18.0.2:48992 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:29,904 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
master-1      | 2025-06-06 13:43:29,905 - INFO - Successfully replicated to http://secondary1:8001
master-1      | INFO:     172.18.0.1:33528 - "POST /messages HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:29,936 - INFO - Received POST request with message_id: 1b119efa-701d-42bc-af45-d8cf24971c3b, content: Msg3, w: 3
master-1      | 2025-06-06 13:43:30,020 - INFO - Received POST request with message_id: 832c978c-0f4c-42c0-ad33-c2eb7ec93b15, content: Msg4, w: 1
master-1      | INFO:     172.18.0.1:33542 - "POST /messages HTTP/1.1" 200 OK
secondary2-1  | INFO:     Started server process [1]
secondary2-1  | INFO:     Waiting for application startup.
secondary2-1  | 2025-06-06 13:43:31,465 - INFO - Starting up, syncing with master
master-1      | 2025-06-06 13:43:31,503 - INFO - Received sync request with last_message_id: None
master-1      | INFO:     172.18.0.4:45006 - "POST /sync HTTP/1.1" 200 OK
secondary2-1  | 2025-06-06 13:43:31,505 - INFO - HTTP Request: POST http://master:8000/sync "HTTP/1.1 200 OK"
secondary2-1  | 2025-06-06 13:43:31,506 - INFO - Synced message: 31b659a4-a651-412b-89ae-8adca3f55403, Msg1, order=1
secondary2-1  | 2025-06-06 13:43:31,507 - INFO - Synced message: 8390c54d-7d04-490f-acbb-bfa2e8205b42, Msg2, order=2
secondary2-1  | 2025-06-06 13:43:31,507 - INFO - Synced message: 1b119efa-701d-42bc-af45-d8cf24971c3b, Msg3, order=3
secondary2-1  | 2025-06-06 13:43:31,508 - INFO - Synced message: 832c978c-0f4c-42c0-ad33-c2eb7ec93b15, Msg4, order=4
secondary2-1  | 2025-06-06 13:43:31,508 - INFO - Sync completed, max_display_order=4
secondary2-1  | 2025-06-06 13:43:31,508 - INFO - Replicated messages after sync: [('31b659a4-a651-412b-89ae-8adca3f55403', 'Msg1', 1), ('8390c54d-7d04-490f-
acbb-bfa2e8205b42', 'Msg2', 2), ('1b119efa-701d-42bc-af45-d8cf24971c3b', 'Msg3', 3), ('832c978c-0f4c-42c0-ad33-c2eb7ec93b15', 'Msg4', 4)]
secondary2-1  | INFO:     Application startup complete.
```



```
secondary2-1  | 2025-06-06 13:43:31,750 - INFO - Received replication request with message_id: 832c978c-0f4c-42c0-ad33-c2eb7ec93b15, content: Msg4, order: 4
secondary1-1  | 2025-06-06 13:43:31,747 - WARNING - Generating random 500 error for message_id: 1b119efa-701d-42bc-af45-d8cf24971c3b
master-1      | 2025-06-06 13:43:31,749 - WARNING - Internal server error from http://secondary1:8001, retrying after 1s
secondary2-1  | 2025-06-06 13:43:31,751 - INFO - Message with message_id: 832c978c-0f4c-42c0-ad33-c2eb7ec93b15 already exists, skipping
secondary1-1  | INFO:     172.18.0.2:49000 - "POST /replicate HTTP/1.1" 500 Internal Server Error
master-1      | 2025-06-06 13:43:31,752 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
secondary2-1  | INFO:     172.18.0.2:56210 - "POST /replicate HTTP/1.1" 200 OK
secondary1-1  | 2025-06-06 13:43:31,748 - INFO - Received replication request with message_id: 832c978c-0f4c-42c0-ad33-c2eb7ec93b15, content: Msg4, order: 4
master-1      | 2025-06-06 13:43:31,753 - INFO - Successfully replicated to http://secondary2:8001
secondary2-1  | 2025-06-06 13:43:32,801 - INFO - Received replication request with message_id: 31b659a4-a651-412b-89ae-8adca3f55403, content: Msg1, order: 4
secondary2-1  | 2025-06-06 13:43:32,802 - INFO - Message with message_id: 31b659a4-a651-412b-89ae-8adca3f55403 already exists, skipping
secondary1-1  | 2025-06-06 13:43:32,802 - INFO - Received replication request with message_id: 1b119efa-701d-42bc-af45-d8cf24971c3b, content: Msg3, order: 4
master-1      | 2025-06-06 13:43:32,803 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
secondary2-1  | INFO:     172.18.0.2:59422 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:32,804 - INFO - Successfully replicated to http://secondary2:8001
secondary1-1  | 2025-06-06 13:43:33,747 - INFO - Message replicated: 832c978c-0f4c-42c0-ad33-c2eb7ec93b15, Msg4, order=4
secondary1-1  | INFO:     172.18.0.2:49004 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:33,749 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
master-1      | 2025-06-06 13:43:33,750 - INFO - Successfully replicated to http://secondary1:8001
master-1      | 2025-06-06 13:43:33,794 - ERROR - Error replicating to http://secondary2:8001: [Errno -2] Name or service not known
secondary1-1  | 2025-06-06 13:43:34,803 - INFO - Message replicated: 1b119efa-701d-42bc-af45-d8cf24971c3b, Msg3, order=4
secondary1-1  | INFO:     172.18.0.2:46024 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:34,823 - INFO - HTTP Request: POST http://secondary1:8001/replicate "HTTP/1.1 200 OK"
master-1      | 2025-06-06 13:43:34,824 - INFO - Successfully replicated to http://secondary1:8001
secondary2-1  | 2025-06-06 13:43:34,853 - INFO - Received replication request with message_id: 8390c54d-7d04-490f-acbb-bfa2e8205b42, content: Msg2, order: 4
secondary2-1  | 2025-06-06 13:43:34,853 - INFO - Message with message_id: 8390c54d-7d04-490f-acbb-bfa2e8205b42 already exists, skipping
master-1      | 2025-06-06 13:43:34,855 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
secondary2-1  | INFO:     172.18.0.2:59432 - "POST /replicate HTTP/1.1" 200 OK
master-1      | 2025-06-06 13:43:34,856 - INFO - Successfully replicated to http://secondary2:8001
secondary2-1  | 2025-06-06 13:43:34,855 - INFO - Received replication request with message_id: 1b119efa-701d-42bc-af45-d8cf24971c3b, content: Msg3, order: 4
master-1      | 2025-06-06 13:43:34,857 - INFO - HTTP Request: POST http://secondary2:8001/replicate "HTTP/1.1 200 OK"
secondary2-1  | 2025-06-06 13:43:34,855 - INFO - Message with message_id: 1b119efa-701d-42bc-af45-d8cf24971c3b already exists, skipping
master-1      | 2025-06-06 13:43:34,858 - INFO - Successfully replicated to http://secondary2:8001
secondary2-1  | INFO:     172.18.0.2:59442 - "POST /replicate HTTP/1.1" 200 OK
master-1      | INFO:     172.18.0.1:33528 - "POST /messages HTTP/1.1" 200 OK
secondary2-1  | 2025-06-06 13:43:44,886 - INFO - Received GET request for replicated messages
secondary2-1  | 2025-06-06 13:43:44,887 - INFO - All replicated messages: [('31b659a4-a651-412b-89ae-8adca3f55403', 'Msg1', 1), ('8390c54d-7d04-490f-acbb-bf
a2e8205b42', 'Msg2', 2), ('1b119efa-701d-42bc-af45-d8cf24971c3b', 'Msg3', 3), ('832c978c-0f4c-42c0-ad33-c2eb7ec93b15', 'Msg4', 4)]
secondary2-1  | 2025-06-06 13:43:44,887 - INFO - Displayed messages: [('31b659a4-a651-412b-89ae-8adca3f55403', 'Msg1', 1), ('8390c54d-7d04-490f-acbb-bfa2e82
05b42', 'Msg2', 2), ('1b119efa-701d-42bc-af45-d8cf24971c3b', 'Msg3', 3), ('832c978c-0f4c-42c0-ad33-c2eb7ec93b15', 'Msg4', 4)]
secondary2-1  | INFO:     172.18.0.1:40660 - "GET /messages HTTP/1.1" 200 OK
```

Логи підтверджують, що secondary2 коректно синхронізував усі повідомлення в test_acceptance.

Тести використовують pytest-asyncio для обробки асинхронних запитів через httpx.AsyncClient. Secondary генерують випадкові HTTP 500 помилки з ймовірністю 20%, що тестує механізм повторних спроб у master.py.

У test_deduplication видно, як Master обробляє повторні запити, повертаючи той самий message_id. test_acceptance перевіряє синхронізацію secondary2 після перезапуску, що включає запит до /sync і отримання всіх повідомлень від Master.

Також кожен тест викликає clear_all, який очищає повідомлення на всіх вузлах через POST /clear, забезпечуючи ізоляцію тестів.