

**Technical Assignment 2**

**Objective**

Create a simplified in-memory **Pub/Sub system** where:

● Publishing and subscribing happen over a **WebSocket endpoint** (/ws).

● Management operations (create/delete/list topics, health, stats) happen via **HTTP REST APIs**.

● The system must handle multiple publishers and subscribers safely,

● No external DBs or brokers (Redis, Kafka, RabbitMQ) may be used — state should be in-memory only.

**Project Scope**

● Implement in **Go**, **Python, or any programming language**.

● WebSocket endpoint (/ws) must support: publish, subscribe, unsubscribe, ping. ● REST APIs must support topic management and observability.

● No persistence across restarts.

● Document assumptions and design choices (e.g., backpressure policy) in README. ● Provide a **Dockerfile** to run the service in a container.

**WebSocket Protocol**

**Client → Server Messages**

JSON

{

"type": "subscribe" | "unsubscribe" | "publish" | "ping",

"topic": "orders", // required for

subscribe/unsubscribe/publish

"message": { // required for publish

"id": "550e8400-e29b-41d4-a716-446655440000",

"payload": "..."

},

"client\_id": "s1", // required for subscribe/unsubscribe



"last\_n": 0, // optional: number of historical messages to replay

"request\_id": "uuid-optional" // optional: correlation id }

**Examples**

**subscribe:**

JSON

{

"type": "subscribe",

"topic": "orders",

"client\_id": "s1",

"last\_n": 5,

"request\_id": "550e8400-e29b-41d4-a716-446655440000"

}

**unsubscribe:**

JSON

{

"type": "unsubscribe",

"topic": "orders",

"client\_id": "s1",

"request\_id": "340e8400-e29b-41d4-a716-4466554480098"

}

**publish:**

JSON

{

"type": "publish",

"topic": "orders",



"message": {

"id": "550e8400-e29b-41d4-a716-446655440000",

"payload": {

"order\_id": "ORD-123",

"amount": "99.5",

"currency": "USD"

}

},

"request\_id": "340e8400-e29b-41d4-a716-4466554480098"

}

**ping:**

JSON

{

"type": "ping",

"request\_id": "570t8400-e29b-41d4-a716-4466554412345"

}

**Server → Client Messages**

JSON

{

"type": "ack" | "event" | "error" | "pong" | "info",

"request\_id": "uuid-optional", // echoed if provided "topic": "orders",

"message": {

"id": "550e8400-e29b-41d4-a716-446655440000",

"payload": "..."

},

"error": {

"code": "BAD\_REQUEST",

"message": "..."

},

"ts": "2025-08-25T10:00:00Z" // optional server timestamp }



**Examples**

**ack** → confirms a successful publish, subscribe, or unsubscribe

JSON

{

"type": "ack",

"request\_id": "550e8400-e29b-41d4-a716-446655440000",

"topic": "orders",

"status": "ok",

"ts": "2025-08-25T10:00:00Z"

}

**event** → a published message delivered to a subscriber (with timestamp)

JSON

{

"type": "event",

"topic": "orders",

"message": {

"id": "550e8400-e29b-41d4-a716-446655440000",

"payload": {

"order\_id": "ORD-123",

"amount": 99.5,

"currency": "USD"

}

},

"ts": "2025-08-25T10:01:00Z"

}

**error** → validation or flow errors

JSON

{

"type": "error",

"request\_id": "req-67890",

"error": {

"code": "BAD\_REQUEST",

"message": "message.id must be a valid UUID"



},

"ts": "2025-08-25T10:02:00Z"

}

Other possible error codes:

● TOPIC\_NOT\_FOUND → publish/subscribe to non-existent topic ● SLOW\_CONSUMER → subscriber queue overflow

● UNAUTHORIZED → invalid/missing auth (if implemented)

● INTERNAL → unexpected server error

**pong** → response to client ping

JSON

{

"type": "pong",

"request\_id": "ping-abc",

"ts": "2025-08-25T10:03:00Z"

}

**info** → server-initiated notice

● Heartbeat

JSON

{

"type": "info",

"msg": "ping",

"ts": "2025-08-25T10:04:00Z"

}

● Topic deleted



JSON

{

"type": "info",

"topic": "orders",

"msg": "topic\_deleted",

"ts": "2025-08-25T10:05:00Z"

}

**HTTP REST Endpoints**

● **POST /topics**

Request: { "name": "orders" }

○ 201 Created → { "status": "created", "topic": "orders" } ○ 409 Conflict if already exists

● **DELETE /topics/{name}**

○ 200 OK → { "status": "deleted", "topic": "orders" }

○ 404 if not found

(Subscribers must be unsubscribed/disconnected)

● **GET /topics**

JSON

{

"topics": [

{

"name": "orders",

"subscribers": 3

}

]

}



● **GET /health**

JSON

{

"uptime\_sec": 123,

"topics": 2,

"subscribers": 4

}

● **GET /stats**

JSON

{

"topics": {

"orders": {

"messages": 42,

"subscribers": 3

}

}

}

**Handling Requirements**

● Concurrency safety for multiple publishers/subscribers.

● Fan-out: every subscriber to a topic receives each message once.

● Isolation: no cross-topic leakage.

● Backpressure: bounded per-subscriber queues. Overflow → drop oldest OR disconnect with error (SLOW\_CONSUMER).

● Graceful shutdown: stop accepting new ops, best-effort flush, close sockets.



**Optional Stretch Goals**

● **Backpressure**: bounded per-subscriber queues; on overflow either drop oldest or disconnect with SLOW\_CONSUMER error (document the policy).

● **Graceful shutdown**: stop accepting new operations, best-effort flush, and close sockets cleanly.

● **Replay**: ring buffer (e.g., last 100 messages) with last\_n support.

● **Basic authentication**: X-API-Key for REST/WS.

**Timeline**

Time-boxed: **2 hours**.

Leverage AI tools like Cursor, Copilot, ChatGPT to fast track the development and complete assignments on time.

**Evaluation Criteria**

● **Correctness (40 pts)** → WebSocket pub/sub works; fan-out & isolation correct; REST matches contract

● **Concurrency & Robustness (20 pts)** → Race-free; stable under multiple clients ● **Code Quality (20 pts)** → Clean structure, naming, error handling

● **Operational Basics (10 pts)** → Heartbeats, config flags, README clarity, Docker run works

● **Polish / Stretch (10 pts)** → last\_n, metrics, backpressure or auth

**Submission**

● Provide a **GitHub repository** with code.

● Include a **README** with setup and **Docker run instructions**.

● Document assumptions (e.g., backpressure policy).