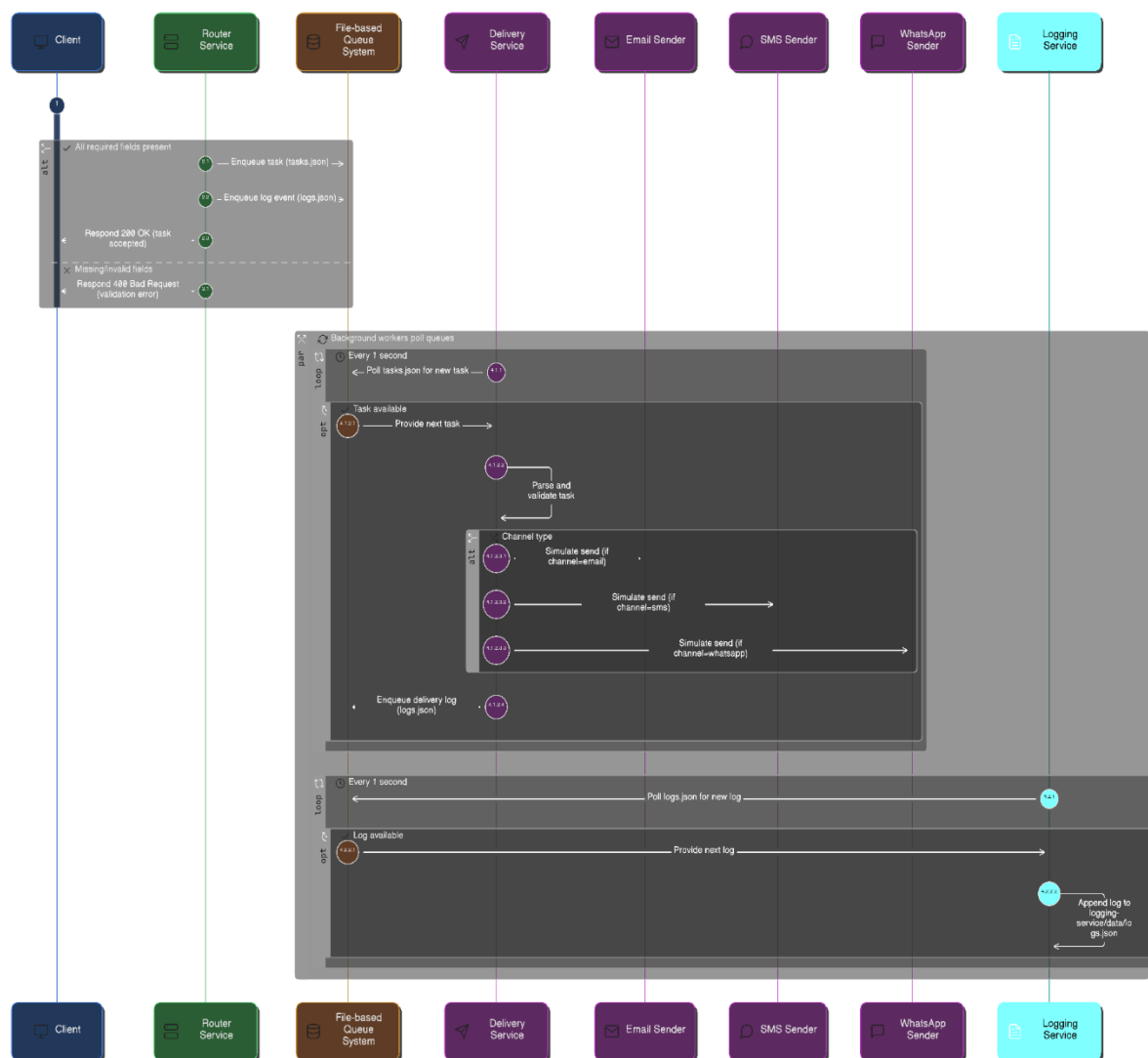


# High Level Design (HLD) — Communication Aggregator

## Overview

A minimal 3-service prototype that routes, delivers (simulated), and logs messages. Services communicate via a simple file-based queue (router-service/data/queues/\*.json) so the project runs locally without external brokers.

## Architecture Diagram



# Components

## Router Service

- Tech: Node.js + Fastify
- Entry point: router-service/src/index.js (HTTP server)
- Port: 4000
- Validates incoming payloads (id, channel, to, body) using src/validators.js.
- Enqueues to tasks and logs using router-service/rabbit.js (file-queue).
- Queue files: router-service/data/queues/tasks.json, router-service/data/queues/logs.json.

## Delivery Service

- Tech: Node.js
- Entry point: delivery-service/src/index.js (background worker)
- Consumes tasks via delivery-service/rabbit.js (re-exports router helper).
- Uses sender stubs in src/senders/ (email.js, sms.js, whatsapp.js) which write sent records to delivery-service/data/ files.
- Writes delivery records to delivery-service/data/deliveries.json (and channel-specific files).

## Logging Service

- Tech: Node.js
- Entry point: logging-service/src/index.js (background worker)
- Consumes logs via logging-service/rabbit.js (re-exports router helper).
- Persists logs to logging-service/data/logs.json.

## Queue helper (rabbit.js)

- Location: router-service/rabbit.js (shared via one-line re-exports in other services).
- API used by services: enqueue(queueName, item) and consume(queueName, handler).
- Implementation: appends items to data/queues/<queue>.json and polls those files every second to deliver items to consumers.

## Data formats (used in code)

- Message (client → router): { id, channel, to, body }
- Task item: { trace, data } where trace is a timestamp string
- Log item: free-form object with event and trace (created by router and delivery)

## Runtime flow (step-by-step)

1. Client POST /message → Router validates payload.
2. Router calls enqueue('tasks', { trace, data }) and enqueue('logs', {...}) and returns success.
3. Delivery service polls tasks.json, processes each task, writes delivery records, and enqueues a delivery log.
4. Logging service polls logs.json and appends entries to logging-service/data/logs.json.

# Run instructions

## 1. Start Router - Service:

```
cd router-service  
npm install  
npm start
```

## 2. Start Delivery - Service:

```
cd delivery-service  
npm install  
npm start
```

## 3. Start Logging - Service:

```
cd logging-service  
npm install  
npm start
```

## 4. Send test message:

### Postman Curl:

```
curl -X POST http://localhost:4000/message -H "Content-Type: application/json" -d  
'{"id":"m1","channel":"email","to":"example@email.com","body":"hello"}'
```